(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 November 2001 (08.11.2001)

PCT

(10) International Publication Number
WO 01/84301 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US01/10605

(22) International Filing Date: 2 April 2001 (02.04.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/563,726 2 May 2000 (02.05.2000) US

(71) Applicant: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052 (US).

(72) Inventors: **SANKARANARAYAN, Mukund**; 25840 SE 41st Street, Issaquah, WA 98029 (US). **FOLTZ, Forrest, C.**; 12450 203rd Avenue NE, Woodinville, WA 98072 (US). **SHAW, George**; 21213 NE 186th Street, Woodinville, WA 98072 (US). **SATHER, Dale, A.**; 7453 4th Avenue NE, Seattle, WA 98115 (US). **RAFFMAN, Andy, R.**; 17835 NE 205th Street, Woodinville, WA 98072 (US). **SRINIVASAN, Jai**; 7512 130th Avenue NE, Kirkland, WA 98033 (US). **BACKMAN, Terje, K.**; P.O. Box 968, Carnation, WA (US). **PARRY, William, G.**; 6731 153rd Place SE, Bellevue, WA 98006 (US). **BAKIN, David, S.**; 1221 First Avenue, Apt. 1501, Seattle, WA 98101 (US). **JONES, Michael, B.**; 17259 NE 36th Street, Redmond, WA 98052 (US). **MCDOWELL, Sean, C.**; 10427 248th Avenue NE, Redmond, WA 98053 (US). **RAJA, Jayachandran**; 6003 143rd Court NE, Redmond, WA 98052 (US). **SPEED, Robin**; 518 2nd AVenue S, Kirkland, WA 98033 (US).

(74) Agents: **LEE, Lewis, C.** et al.; 421 W. Riverside Avenue, Suite 500, Spokane, WA 99201 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/84301 A2

(54) Title: RESOURCE MANAGER ARCHITECTURE

(57) Abstract: Resource management architectures implemented in computer systems to manage resources are described. In one embodiment, a general architecture includes a resource manager and multiple resource providers that support one or more resource consumers such as a system component or application. Each provider is associated with a resource and acts as the manager for the resource when interfacing with the resource manager. The resource manager arbitrates access to the resources provided by the resource providers on behalf of the consumers. A policy manager sets various policies that are used by the resource manager to allocate resources. One policy is a priority-based policy that distinguishes among which applications and/or users have priority over others to use the resources. A resource consumer creates an "activity" at the resource manager and builds one or more "configurations" that describe various sets of preferred resources required to perform the activity. Each resource consumer can specify one or more configurations for each activity. If multiple configurations are specified, the resource consumer can rank them according to preference. This allows the resource consumers to be dynamically changed from one configuration to another as operating conditions change.

(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-508611
(P2004-508611A)
(43)公表日　平成16年3月18日(2004.3.18)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 9/46	G06F	9/46　340H	5B098

審査請求 未請求　予備審査請求 有　（全 212 頁）

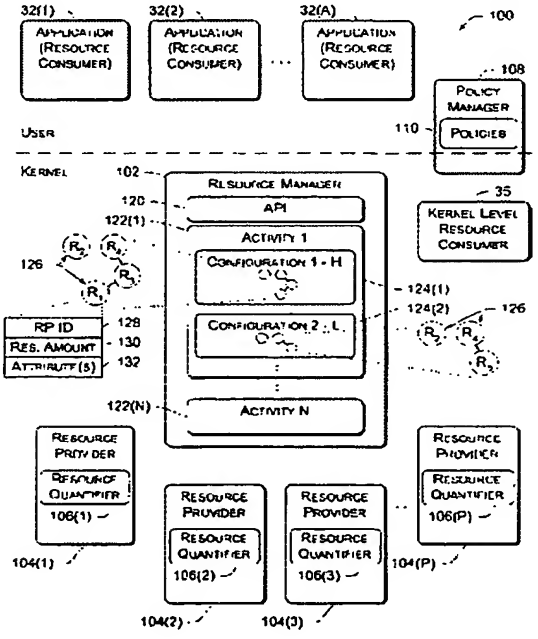
(21)出願番号　特願2001-580657(P2001-580657)
(86)(22)出願日　平成13年4月2日(2001.4.2)
(85)翻訳文提出日　平成14年11月5日(2002.11.5)
(86)国際出願番号　PCT/US2001/010605
(87)国際公開番号　WO2001/084301
(87)国際公開日　平成13年11月8日(2001.11.8)
(31)優先権主張番号　09/563,726
(32)優先日　平成12年5月2日(2000.5.2)
(33)優先権主張国　米国(US)

(71)出願人　391055933
マイクロソフト　コーポレイション
ＭＩＣＲＯＳＯＦＴ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ　（番地なし）
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(74)代理人　100106998
弁理士　橋本　傳一

最終頁に続く

(54)【発明の名称】リソースマネージャアーキテクチャ

(57)【要約】
リソースを管理するためにコンピュータシステム中で実現されるリソース管理アーキテクチャについて述べる。一実施形態では、一般的なアーキテクチャは、リソースマネージャと、システムコンポーネントやアプリケーションなどの１つ以上のリソースコンシューマをサポートする複数のリソースプロバイダとを含む。各プロバイダは１つのリソースに関連し、リソースマネージャとインタフェースしているときにこのリソースのためのマネージャとして行動する。リソースマネージャは、コンシューマのために、リソースプロバイダから提供されるリソースへのアクセスを調停する。ポリシーマネージャが、リソースを割り振るためにリソースマネージャによって使用される様々なポリシーを設定する。あるポリシーは、どのアプリケーションおよび／またはユーザが他を制してリソース使用の優先度を有するかを識別する、優先度に基づくポリシーである。リソースコンシューマは、リソースマネージャに「アクティビティ」を生み出し、アクティビティを実施するのに必要な好ましいリソースの様々なセットを記述する１つ以上の「構成」を構築す



【特許請求の範囲】
【請求項１】
リソース管理アーキテクチャであって、
リソースに関連する複数のリソースプロバイダと、
前記リソースプロバイダから提供されるリソースを利用する複数のコンシューマと、
前記リソースプロバイダから提供されるリソースへのアクセスを、前記コンシューマのために調停するリソースマネージャとを備えることを特徴とするリソース管理アーキテクチャ。
【請求項２】
請求項１に記載のリソース管理アーキテクチャであって、前記リソースプロバイダおよび前記リソースマネージャがカーネルレベルに存在することを特徴とするリソース管理アーキテクチャ。
【請求項３】
請求項１に記載のリソース管理アーキテクチャであって、前記コンシューマのうちの少なくとも１つがアプリケーションプログラムを備えることを特徴とするリソース管理アーキテクチャ。
【請求項４】
請求項１に記載のリソース管理アーキテクチャであって、少なくとも１つのリソースプロバイダが、少なくとも１つの他のリソースプロバイダから提供されるリソースのコンシューマとして行動することを特徴とするリソース管理アーキテクチャ。
【請求項５】
請求項１に記載のリソース管理アーキテクチャであって、前記コンシューマが、前記リソースを予約する要求を前記リソースマネージャに提出することを特徴とするリソース管理アーキテクチャ。
【請求項６】
請求項５に記載のリソース管理アーキテクチャであって、前記リソースプロバイダが、前記コンシューマが前記リソースを最初に予約せずに使用することができないようにすることを特徴とするリソース管理アーキテクチャ。
【請求項７】
請求項１に記載のリソース管理アーキテクチャであって、前記リソースプロバイダが前記リソースマネージャに登録することを特徴とするリソース管理アーキテクチャ。
【請求項８】
請求項１に記載のリソース管理アーキテクチャであって、前記リソースプロバイダの各々が、割振りに利用可能な関連するリソースの量を決定するリソース定量化部を備えることを特徴とするリソース管理アーキテクチャ。
【請求項９】
請求項８に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、割振りにいくつのリソースが利用可能かについてのカウントを維持するカウンタを備えることを特徴とするリソース管理アーキテクチャ。
【請求項１０】
請求項８に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、リソースを割振りに利用できる期間を追跡する時間追跡コンポーネントを備えることを特徴とするリソース管理アーキテクチャ。
【請求項１１】
請求項８に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、割振りに利用可能なリソースのパーセンテージを追跡するパーセンテージ追跡コンポーネントを備えることを特徴とするリソース管理アーキテクチャ。
【請求項１２】
請求項１に記載のリソース管理アーキテクチャであって、前記リソースマネージャがアプリケーションプログラムインタフェースを公開し、前記リソースプロバイダおよび前記コ

ンシューマが、前記アプリケーションプログラムインタフェースを介して前記リソースマネージャへの呼出しを行うことを特徴とするリソース管理アーキテクチャ。

【請求項１３】

請求項１に記載のリソース管理アーキテクチャであって、前記リソースマネージャが、優先度のより高いコンシューマを優先度のより低いコンシューマよりも優先する優先度ベースのポリシーに従って、前記リソースを前記コンシューマに割り振ることを特徴とするリソース管理アーキテクチャ。

【請求項１４】

請求項１に記載のリソース管理アーキテクチャであって、前記コンシューマが前記リソースマネージャにおいてアクティビティを生み出し、各アクティビティが、対応するコンシューマによって実施されるタスクと関連することを特徴とするリソース管理アーキテクチャ。

【請求項１５】

請求項１４に記載のリソース管理アーキテクチャであって、前記コンシューマが、前記アクティビティの各々につき１つ以上の構成を前記リソースマネージャにおいて構築し、各構成が、前記タスクを実施する１つ以上のリソースのセットを識別することを特徴とするリソース管理アーキテクチャ。

【請求項１６】

請求項１５に記載のリソース管理アーキテクチャであって、前記各構成が、１つ以上のリソース記述子のセットを含み、各記述子が、対応するリソースのインスタンスであることを特徴とするリソース管理アーキテクチャ。

【請求項１７】

請求項１６に記載のリソース管理アーキテクチャであって、前記各記述子が、前記リソースに関連するリソースプロバイダの識別と、前記タスクを実施するのに必要な前記リソースの量とを含むデータを保持することを特徴とするリソース管理アーキテクチャ。

【請求項１８】

請求項１に記載のリソース管理アーキテクチャであって、

前記コンシューマのうちの１つが、実施するタスクに関連して前記リソースマネージャにおいてアクティビティを生み出し、前記アクティビティについて複数の構成を構築し、各構成が、前記タスクを実施する１つ以上のリソースのセットを識別し、

前記リソースマネージャが、特定のリソースを前記アクティビティに再割振りするかまたは前記アクティビティから取り去って再割振りするために、前記アクティビティを前記構成のうちの１つから前記構成のうちの別の１つに切り替えるように構成されていることを特徴とするリソース管理アーキテクチャ。

【請求項１９】

請求項１８に記載のリソース管理アーキテクチャであって、前記リソースマネージャが、前記１つの構成から前記別の１つの構成に自動的に切り替えるように構成されていることを特徴とするリソース管理アーキテクチャ。

【請求項２０】

請求項１８に記載のリソース管理アーキテクチャであって、前記構成が好ましさの順にランク付けされ、前記リソースマネージャが、前記アクティビティをより好ましい構成からより好ましくない構成に切り替えるように構成されていることを特徴とするリソース管理アーキテクチャ。

【請求項２１】

請求項１に記載のリソース管理アーキテクチャであって、

前記コンシューマのうちの１つが、実施するタスクに関連して前記リソースマネージャにおいてアクティビティを生み出し、前記アクティビティについて複数の構成を構築し、各構成が、前記タスクを実施する１つ以上のリソースのセットを識別し、

前記リソースマネージャが、前記構成のうちの少なくとも１つを満たすリソースを予約するように構成されていることを特徴とするリソース管理アーキテクチャ。

【請求項２２】
請求項２１に記載のリソース管理アーキテクチャであって、前記構成が好ましさの順にランク付けされ、予約されたリソースを有する構成が構成のうちで最高ランクではなく、前記リソースマネージャが、より高ランクの構成のためのリソースが利用可能になったときに、前記１つのコンシューマにアップグレード通知を送ることを特徴とするリソース管理アーキテクチャ。
【請求項２３】
請求項２２に記載のリソース管理アーキテクチャであって、前記１つのコンシューマが、前記アップグレード通知に応答して、前記リソースマネージャに、前記より高ランクの構成のためのリソースを予約するように要求することを特徴とするリソース管理アーキテクチャ。
【請求項２４】
請求項１に記載のリソース管理アーキテクチャであって、割振りポリシーを確立するポリシーマネージャをさらに備え、前記リソースマネージャが前記割振りポリシーを使用して前記リソースへのアクセスを調停することを特徴とするリソース管理アーキテクチャ。
【請求項２５】
請求項１に記載のリソース管理アーキテクチャであって、前記コンシューマのために１つ以上のリソースのセットを要求するインタフェースコンポーネントをさらに備え、前記インタフェースコンポーネントが前記リソースマネージャに要求を提出することを特徴とするリソース管理アーキテクチャ。
【請求項２６】
請求項１に記載のリソース管理アーキテクチャであって、前記リソースプロバイダがリアルタイムの知識を全く有さないように構成され、後の時点における前記リソースのセットの割振りを予定するスケジューリングコンポーネントをさらに備えることを特徴とするリソース管理アーキテクチャ。
【請求項２７】
リソース管理アーキテクチャであって、
リソースに関連する複数のリソースプロバイダと、
タスクを実施するために、前記リソースプロバイダから提供されるリソースを利用する複数のコンシューマと、
前記リソースプロバイダから提供されるリソースを、前記コンシューマのうちの１つ以上に割り振るためのリソースマネージャと、
対応するコンシューマによって実施されるタスクに関連して前記リソースマネージャにおいて生み出される少なくとも１つのアクティビティデータ構造と、
前記アクティビティデータ構造に関連して前記リソースマネージャにおいて生み出される少なくとも１つの構成データ構造であって、関連するアクティビティを実施するのに使用される１つ以上のリソースのセットを記述する構成データ構造とを備えることを特徴とするリソース管理アーキテクチャ。
【請求項２８】
請求項２７に記載のリソース管理アーキテクチャであって、前記構成データ構造が前記アクティビティデータ構造内に含まれることを特徴とするリソース管理アーキテクチャ。
【請求項２９】
請求項２７に記載のリソース管理アーキテクチャであって、前記構成データ構造が、前記リソースプロバイダのうちの１つを識別するリソース記述子を少なくとも１つ含むことを特徴とするリソース管理アーキテクチャ。
【請求項３０】
請求項２７に記載のリソース管理アーキテクチャであって、前記リソースプロバイダおよび前記リソースマネージャがカーネルレベルに存在することを特徴とするリソース管理アーキテクチャ。
【請求項３１】

請求項２７に記載のリソース管理アーキテクチャであって、前記コンシューマのうちの少なくとも１つがアプリケーションプログラムを備えることを特徴とするリソース管理アーキテクチャ。

【請求項３２】

請求項２７に記載のリソース管理アーキテクチャであって、少なくとも１つのリソースプロバイダが、少なくとも１つの他のリソースプロバイダから提供されるリソースのコンシューマとして行動することを特徴とするリソース管理アーキテクチャ。

【請求項３３】

請求項２７に記載のリソース管理アーキテクチャであって、前記リソースプロバイダが前記リソースマネージャに登録することを特徴とするリソース管理アーキテクチャ。

【請求項３４】

請求項２７に記載のリソース管理アーキテクチャであって、前記リソースプロバイダの各々が、割振りに利用可能な関連するリソースの量を決定するリソース定量化部を備えることを特徴とするリソース管理アーキテクチャ。

【請求項３５】

請求項３４に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、割振りにいくつのリソースが利用可能かについてのカウントを維持するカウンタを備えることを特徴とするリソース管理アーキテクチャ。

【請求項３６】

請求項３４に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、リソースを割振りに利用できる期間を追跡する時間追跡コンポーネントを備えることを特徴とするリソース管理アーキテクチャ。

【請求項３７】

請求項３４に記載のリソース管理アーキテクチャであって、前記リソース定量化部が、割振りに利用可能なリソースのパーセンテージを追跡するパーセンテージ追跡コンポーネントを備えることを特徴とするリソース管理アーキテクチャ。

【請求項３８】

請求項２７に記載のリソース管理アーキテクチャであって、前記リソースマネージャがアプリケーションプログラムインタフェースを公開し、前記リソースプロバイダおよび前記コンシューマが、前記アプリケーションプログラムインタフェースを介して前記リソースマネージャへの呼出しを行うことを特徴とするリソース管理アーキテクチャ。

【請求項３９】

請求項２７に記載のリソース管理アーキテクチャであって、前記リソースマネージャが、優先度のより高いコンシューマを優先度のより低いコンシューマよりも優先する優先度ベースのポリシーに従って、前記リソースを前記コンシューマに割り振ることを特徴とするリソース管理アーキテクチャ。

【請求項４０】

請求項２７に記載のリソース管理アーキテクチャであって、前記アクティビティデータ構造に関連して生み出された複数の構成データ構造をさらに備えることを特徴とするリソース管理アーキテクチャ。

【請求項４１】

請求項２７に記載のリソース管理アーキテクチャであって、前記リソースマネージャが、前記構成データ構造のうちの１つの中で指定されるリソースを割り振ることを特徴とするリソース管理アーキテクチャ。

【請求項４２】

請求項４１に記載のリソース管理アーキテクチャであって、前記リソースマネージャが、前記１つの構成データ構造中で指定されるリソースのうちの少なくとも１つを後で再割振りし、前記構成データ構造のうちの別の１つの中で指定される異なるリソースセットを自動的に予約することを特徴とするリソース管理アーキテクチャ。

【請求項４３】

請求項４２に記載のリソース管理アーキテクチャであって、前記１つの構成データ構造中で指定されるリソースが再び利用可能になったとき、前記リソースマネージャが、前記アクティビティデータ構造に対応するコンシューマに通知することを特徴とするリソース管理アーキテクチャ。

【請求項４４】

請求項４３に記載のリソース管理アーキテクチャであって、前記コンシューマが、前記通知に応答して、前記リソースマネージャに、前記１つの構成データ構造中で指定されるリソースを予約するように要求することを特徴とするリソース管理アーキテクチャ。

【請求項４５】

リソース管理アーキテクチャであって、

リソースに関連するリソースプロバイダと、

少なくとも部分的に前記リソースによって実施されるタスクに関連してアクティビティを生み出し、前記アクティビティについて、前記リソースを識別する少なくとも１つの構成を構築するリソースマネージャであって、さらに、前記タスクを実施するために、前記アクティビティについて、前記構成内で識別されるリソースを予約することができるかどうかを決定するように構成されたリソースマネージャとを備えることを特徴とするリソース管理アーキテクチャ。

【請求項４６】

請求項４５に記載のリソース管理アーキテクチャであって、前記リソースを予約することができるかどうかを決定するために前記リソースマネージャによって使用されるポリシーを確立するポリシーマネージャをさらに備えることを特徴とするリソース管理アーキテクチャ。

【請求項４７】

請求項４５に記載のリソース管理アーキテクチャであって、前記リソースプロバイダが、割振りに利用可能なリソースの量を決定するリソース定量化部を備えることを特徴とするリソース管理アーキテクチャ。

【請求項４８】

請求項４５に記載のリソース管理アーキテクチャであって、前記リソースマネージャがアプリケーションプログラムインタフェースを公開し、前記リソースプロバイダが前記アプリケーションプログラムインタフェースを介して前記リソースマネージャへの呼出しを行うことを特徴とするリソース管理アーキテクチャ。

【請求項４９】

リソースマネージャであって、

タスクを実施するのに使用される１つ以上のリソースを予約する呼出しを可能にするアプリケーションプログラムインタフェースと、

前記タスクに関連する少なくとも１つのアクティビティと、

前記タスクを実施するのに使用される１つ以上のリソースを識別するための、前記アクティビティに関連する少なくとも１つの構成とを備えることを特徴とするリソースマネージャ。

【請求項５０】

請求項４９に記載のリソースマネージャであって、前記構成内で指定される１つ以上のリソースを予約するかどうかを決定する手段をさらに備えることを特徴とするリソースマネージャ。

【請求項５１】

リソースマネージャであって、

リソースをリソースマネージャに登録する手段と、

前記リソースのコンシューマから、関連するタスクを実施するためのリソースのセットを予約する要求を受け取る手段と、

前記要求を満たすことができるかどうかを評価する手段と、

前記要求を満たすことができる場合に、前記コンシューマのために前記リソースのセット

を予約する手段と、
前記要求のすべてを満たすことができない場合に、前記要求のうちのどれを満たすことができるかを決定し、それらの要求を提出したコンシューマのために前記リソースのセットを予約する手段とを備えることを特徴とするリソースマネージャ。
【請求項５２】
リソースマネージャであって、
タスクを実施するためのリソースのセットを予約する要求を受け取る手段と、
実施する前記タスクに関連して新しいアクティビティを生み出す手段と、
前記新しいアクティビティについて、前記リソースのセットを識別する少なくとも１つの新しい構成を構築する手段と、
前記タスクを実施するために、前記新しい構成内で識別される前記リソースのセットを前記新しいアクティビティについて予約することができるかどうかを決定する手段とを備えることを特徴とするリソースマネージャ。
【請求項５３】
方法であって、
リソースをリソースマネージャに登録するステップと、
前記リソースのコンシューマから、関連するタスクを実施するためのリソースのセットを予約する要求を受け取るステップと、
前記要求を満たすことができるかどうかを評価するステップと、
前記要求を満たすことができる場合に、前記コンシューマのために前記リソースのセットを予約するステップと、
前記要求のすべてを満たすことができない場合に、前記要求のうちのどれを満たすことができるかを決定し、それらの要求を提出したコンシューマのために前記リソースのセットを予約するステップとを備えることを特徴とする方法。
【請求項５４】
請求項５３に記載の方法であって、前記評価するステップが、優先度ベースのポリシーに従って、前記リソースを前記コンシューマに割り振ることができるかどうかを決定するステップを備えることを特徴とする方法。
【請求項５５】
請求項５３に記載の方法であって、要求を満たすことができなかったコンシューマに、それらの予約要求が失敗したことを通知するステップをさらに備えることを特徴とする方法。
【請求項５６】
請求項５３に記載の方法であって、コンシューマが前記リソースを最初に予約せずに使用することができないようにするステップをさらに備えることを特徴とする方法。
【請求項５７】
請求項５３に記載の方法であって、前記リソースマネージャにおいて、タスクを実施するのに利用することのできるリソースの複数の代替構成を確立するステップをさらに備えることを特徴とする方法。
【請求項５８】
請求項５７に記載の方法であって、前記タスクを実施するために、ある構成から別の構成に切り替えるステップをさらに備えることを特徴とする方法。
【請求項５９】
請求項５７に記載の方法であって、前記タスクの処理を中断することなく、前記タスクを実施するために、ある構成から別の構成に動的に切り替えるステップをさらに備えることを特徴とする方法。
【請求項６０】
請求項５７に記載の方法であって、
前記構成を好ましさの順にランク付けするステップと、
前記タスクを実施するために、より好ましい構成からより好ましくない構成にダウングレ

ードするステップとをさらに備えることを特徴とする方法。
【請求項６１】
コンピュータ可読媒体であって、コンピュータ実行可能命令を備え、前記コンピュータ実行可能命令は、実行されると、コンピュータシステムに、請求項５３に記載の方法を実施するように命令することを特徴とするコンピュータ可読媒体。
【請求項６２】
方法であって、
タスクを実施するためのリソースのセットを予約する要求を受け取るステップと、
実施する前記タスクに関連して新しいアクティビティを生み出すステップと、
前記新しいアクティビティについて、前記リソースのセットを識別する少なくとも１つの新しい構成を構築するステップと、
前記タスクを実施するために、前記新しい構成内で識別される前記リソースのセットを前記新しいアクティビティについて予約することができるかどうかを決定するステップとを備えることを特徴とする方法。
【請求項６３】
請求項６２に記載の方法であって、前記決定するステップが、
要求されている前記リソースを現在使用しているすべての既存のアクティビティを識別するステップと、
割振りポリシーに従って、前記既存のアクティビティおよび前記新しいアクティビティの間で前記リソースを割り当てるステップとを備えることを特徴とする方法。
【請求項６４】
請求項６３に記載の方法であって、前記ポリシーが優先度ベースのポリシーであり、前記リソースが前記アクティビティに、それらの優先度に従って割り当てられることを特徴とする方法。
【請求項６５】
請求項６２に記載の方法であって、前記決定するステップが、
予約されたリソースを有するすべての既存のアクティビティ、および生み出される前記新しいアクティビティのアクティビティリストを構築するステップと、
前記新しい構成内で識別され、前記アクティビティリスト中の前記既存のアクティビティのために現在予約されているリソースのリソースリストを構築するステップと、
前記アクティビティリスト中のアクティビティによって使用される、前記リソースリスト中のリソースの量を決定するステップと、
前記リソースが前記アクティビティリスト中のすべてのアクティビティを満たすのに不十分な場合、不十分なリソースがあるすべてのアクティビティを識別するステップとを備えることを特徴とする方法。
【請求項６６】
請求項６５に記載の方法であって、前記リソースリストを優先度に従って順序付けるステップをさらに備えることを特徴とする方法。
【請求項６７】
請求項６５に記載の方法であって、前記識別されたアクティビティに、前記リソースが予約できなかったことを通知するステップをさらに備えることを特徴とする方法。
【請求項６８】
請求項６５に記載の方法であって、前記リソースが前記アクティビティリスト中のすべてのアクティビティを満たすのに十分である場合、前記リソースを前記新しいアクティビティのために予約するステップをさらに備えることを特徴とする方法。
【請求項６９】
コンピュータ可読媒体であって、コンピュータ実行可能命令を備え、前記コンピュータ実行可能命令は、実行されると、コンピュータシステムに、請求項６２に記載の方法を実施するように命令することを特徴とするコンピュータ可読媒体。
【請求項７０】

複数のリソースを有するコンピューティングシステムにおいて、新しいアクティビティのためにリソースのセットを予約することができるかどうかを決定する方法であって、
予約されたリソースを有するすべての既存のアクティビティ、および生み出される前記新しいアクティビティのアクティビティリストを構築するステップと、
新しい構成内で識別され、前記アクティビティリスト中の前記既存のアクティビティのために現在予約されているリソースのリソースリストを構築するステップと、
前記アクティビティリスト中のアクティビティによって使用される、前記リソースリスト中のリソースの量を決定するステップと、
前記リソースが前記アクティビティリスト中のすべてのアクティビティを満たすのに不十分な場合、不十分なリソースがあるすべてのアクティビティを識別するステップとを備えることを特徴とする方法。

【請求項７１】
請求項７０に記載の方法であって、前記リソースリストを優先度に従って順序付けるステップをさらに備えることを特徴とする方法。

【請求項７２】
請求項７０に記載の方法であって、前記識別されたアクティビティに、前記リソースが予約できなかったことを通知するステップをさらに備えることを特徴とする方法。

【請求項７３】
請求項７０に記載の方法であって、前記リソースが前記アクティビティリスト中のすべてのアクティビティを満たすのに十分である場合、前記リソースを前記新しいアクティビティのために予約するステップをさらに備えることを特徴とする方法。

【請求項７４】
コンピュータ可読媒体であって、コンピュータ実行可能命令を備え、前記コンピュータ実行可能命令は、実行されると、コンピュータシステムに、請求項７０に記載の方法を実施するように命令することを特徴とするコンピュータ可読媒体。

【請求項７５】
コンピュータ可読媒体であって、コンピュータ実行可能命令を有し、前記コンピュータ実行可能命令は、実行されると、コンピューティングデバイスに、
関連するタスクを実施するためのリソースのセットを予約する要求を受け取り、
前記要求を満たすことができるかどうかを評価し、
前記要求を満たすことができる場合に、前記リソースのセットを予約し、
前記要求のすべてを満たすことができない場合に、前記要求のうちのどれを満たすことができるかを決定し、前記要求を満たすために前記リソースのセットを予約するように命令することを特徴とするコンピュータ可読媒体。

【請求項７６】
請求項７５に記載のコンピュータ可読媒体に組み入れられ、前記コンピュータ実行可能命令を備えることを特徴とするオペレーティングシステム。

【請求項７７】
コンピュータ可読媒体であって、コンピュータ実行可能命令を有し、前記コンピュータ実行可能命令は、実行されると、コンピューティングデバイスに、
タスクを実施するためのリソースのセットを予約する要求を受け取り、
実施する前記タスクに関連して新しいアクティビティを生み出し、
前記新しいアクティビティについて、前記リソースのセットを識別する少なくとも１つの新しい構成を構築し、
要求されている前記リソースを現在使用しているすべての既存のアクティビティを識別し、
割振りポリシーに従って、前記既存のアクティビティおよび前記新しいアクティビティの間で前記リソースを割り当てるように命令することを特徴とするコンピュータ可読媒体。

【請求項７８】
請求項７７に記載のコンピュータ可読媒体であって、前記ポリシーが優先度ベースのポリ

シーであり、前記リソースが前記アクティビティに、それらの優先度に従って割り当てられることを特徴とするコンピュータ可読媒体。

【請求項７９】

請求項７７に記載のコンピュータ可読媒体に組み入れられ、前記コンピュータ実行可能命令を備えることを特徴とするオペレーティングシステム。

【請求項８０】

複数のリソースを管理するリソース管理アーキテクチャを有するコンピュータシステムにおいて、コンピュータ可読媒体に記憶された複数のデータ構造であって、

コンテナとして具体化されたアクティビティデータ構造と、

前記アクティビティデータ構造に含まれ、コンテナとして具体化された少なくとも１つの構成データ構造と、

前記構成データ構造に含まれる少なくとも１つの記述子データ構造とを備え、前記記述子データ構造が、

（ａ）タスクを実施するのに使用されるリソースの識別を保持する第１のデータフィールドと、

（ｂ）前記タスクを実施するのに必要なリソースの量を保持する第２のデータフィールドとを備えることを特徴とするデータ構造。

【請求項８１】

関数呼出しのセットを提供する、コンピュータ可読媒体に組み入れられたアプリケーションプログラムインタフェースであって、前記関数呼出しは、コンピュータ上で実行されると、

コンピュータシステムの一部であるリソースを登録する関数と、

アクティビティオブジェクトを生み出す関数と、

前記アクティビティオブジェクトに関連して構成オブジェクトを生み出す関数と、

前記構成オブジェクトにリソースを加える関数と、

前記構成オブジェクトからリソースを除去する関数と、

前記アクティビティオブジェクトのために前記構成オブジェクト内のリソースのうちの１つ以上を予約する関数とを実施することを特徴とするアプリケーションプログラムインタフェース。

【請求項８２】

複数のリソースと前記リソースを管理するリソースマネージャとを有するコンピュータシステムにおいて、前記リソースマネージャによって公開される、コンピュータ可読媒体に組み入れられたアプリケーションプログラムインタフェースであって、

（１）関連するリソースを提供するように構成されたリソースプロバイダによって呼び出されるリソースプロバイダ関数呼出しであって、コンピュータ上で実行されると、

前記リソースを前記リソースマネージャに登録する関数と、

前記リソースを登録解除する関数と、

前記リソースプロバイダの内部状態を修正できるように前記リソースをロックする関数と、

前記リソースをロック解除する関数と、

リソースを構成オブジェクトに加える関数と、

リソースを前記構成オブジェクトから除去する関数と、

前記リソースの属性を変更する関数と、

前記リソースの属性を取り出す関数とを実施するリソースプロバイダ関数呼出しと、

（２）前記リソースのコンシューマによって呼び出されるコンシューマ関数呼出しであって、コンピュータ上で実行されると、

アクティビティオブジェクトを生み出す関数と、

前記アクティビティオブジェクトのために前記リソースのうちの１つ以上を予約する関数と、

前記リソースを予約解除する関数と、

前記アクティビティオブジェクトの状態を返す関数と、
前記構成オブジェクトを生み出す関数と、
前記構成オブジェクトを除去する関数と、
前記リソースに関する情報を取り出す関数とを実施するコンシューマ関数呼出しとを備えることを特徴とするアプリケーションプログラムインタフェース。
【発明の詳細な説明】
【０００１】
（技術分野）
本発明はコンピュータに関し、より詳細には、コンピュータのリソースを管理するためのシステムおよび方法に関する。
【０００２】
（発明の背景）
コンピュータは、その従来のデスクトップルーツをはるかに超えて進化している。今日のパーソナルコンピュータ（ＰＣ）には、従来のデスクトップアプリケーション（例えばワードプロセッシング、表計算、電子メールなど）に加えて、オーディオビデオファイルの再生、音楽ＣＤ（コンパクトディスク）の再生、ブロードキャスト番組の受信および表示などが求められている。この進化の多くの部分は、計算、インターネット、テレフォニー、および娯楽の技術が不断に収束することによって促進されている。
【０００３】
この結果、コンピュータのルック、フィール、および機能は、様々な消費者環境および動作環境に向けて進化し続けている。例えば、家庭娯楽向けに設計されるコンピュータは、ブラウザソフトウェア、１つ以上のチューナ、ＥＰＧ（電子番組ガイド）ソフトウェア、種々のオーディオ／ビデオドライバ、ゲーミングソフトウェアを搭載した、セットトップボックスまたはゲームコンソールとして実現することができる。オフィス向けに設計されるコンピュータは、従来のデスクトップＰＣに似た外観とすることができるが、より向上したコンピューティング体験を提供するために、ブロードキャストチューナ、ＤＶＤ（ディジタルビデオディスク）ドライバ、サラウンドサウンド付きステレオスピーカなどが付属したものとすることができる。携帯型コンピュータの多様性および機能は、移動ユーザの需要が増加するのに伴って一層広い範囲に及んでいる。
【０００４】
より多様なタスクを実行することがコンピュータに求められている中で、ユーザが複数タスクの同時実施を期待するのは珍しいことではない。このユーザ需要の増加により、様々なタスクに対処するために既存のリソースに対してより多くの需要が生じている。残念ながら、このことにより、すべてのタスクを同時に達成するのに十分なリソースを要求時にコンピュータが備えていない可能性がより高くなる。
【０００５】
このリソース不足はおそらく、家庭娯楽環境向けに設計されたコンピュータの場合に最も明白である。このようなコンピュータは、複数の機能を同時に実施できなければならないだけでなく、異なる複数のユーザの要求も満たさなければならない。例えば、あるユーザは、娯楽コンピュータが指定時間にある番組を録画するように要求し、別のユーザは、コンピュータが同じ時間に異なる番組に合わせるように要求する。コンピュータにチューナが１つしかない場合、両方のタスクを同時に達成することはどうしてもできないので、これは問題である。
【０００６】
このような状況では、コンピュータは、どのタスクを実施すべきであってどのタスクを実施すべきでないかの識別に窮する。今日、アプリケーションは、先着順または遅い順にリソースを得る。したがってアプリケーションは、ユーザの希望に関係なくリソース割振りを制御する。先の例で、テレビジョンアプリケーションがレコーダアプリケーションに勝ってチューナ制御権を握った場合、ユーザが第２の番組を見ることよりも第１の番組を録画することの方にずっと興味があるかもしれなくても、テレビジョンアプリケーションが

リソース（すなわちチューナ）を制御することになる。アプリケーションがリソースを得ると、リソースは、アプリケーションが明示的に明け渡すまでアプリケーションによって保持される。

【０００７】

したがって、リソースに対する需要が増加し続けるのに伴い、リソースと、異なるユーザ／アプリケーションへのリソース割振りとを管理する技術に対する必要がより高まっている。

【０００８】

（発明の概要）

リソースを管理するためにコンピュータシステム中で実現されるリソース管理アーキテクチャについて述べる。

【０００９】

説明する実装形態では、一般的なアーキテクチャは、リソースマネージャと、システムコンポーネントやアプリケーションなどの１つ以上のリソースコンシューマをサポートする複数のリソースプロバイダとを含む。各プロバイダは１つのリソースに関連し、リソースマネージャとインタフェースしているときにこのリソースのためのマネージャとして行動する。リソースマネージャは、コンシューマのために、リソースプロバイダから提供されるリソースへのアクセスを調停する。

【００１０】

このアーキテクチャには、リソースマネージャがリソースを割り振るのに使用する様々なポリシーを設定するためのポリシーマネージャを任意選択で含めることができる。リソースマネージャが使用することのできるポリシーの１つは、どのアプリケーションおよび／またはユーザが他を制してリソース使用の優先度を有するかを決定するための、優先度に基づくポリシーである。

【００１１】

説明する実施形態では、各リソースプロバイダはリソースマネージャに登録する。リソースコンシューマは、リソースマネージャに「アクティビティ」を生み出し、アクティビティを実施するのに必要な様々なリソースセットを記述する１つ以上の「構成」を構築する。アクティビティは、構成を保持するコンテナデータ構造（ｃｏｎｔａｉｎｅｒ ｄａｔａ ｓｔｒｕｃｔｕｒｅ）として実装され、各構成は、リソースの識別を含むデータ構造として実装される。リソースマネージャは、アクティビティおよび構成を維持する。

【００１２】

説明する実施形態では、各リソースコンシューマは、各アクティビティにつき１つ以上の構成を指定することができる。複数の構成を指定する場合、リソースコンシューマは、好ましさ（ｐｒｅｆｅｒｅｎｃｅ）に従ってこれらをランク付けすることができる。これにより、動作条件の変化に応じて、リソースコンシューマをある構成から別の構成に動的に変更することができる。一態様では、優先度のより高いリソースコンシューマがどこかで必要としているリソースがある場合に、現在のリソースコンシューマに、より好ましくない構成を使用するように求めるか、あるいはそのリソース構成をすべて放棄するか必要とされる特定のリソースを放棄するように求めることにより、このリソースを確保することができる。後でこれらのリソースが再び利用可能になったときは、リソースマネージャはリソースコンシューマに通知することができ、したがってリソースコンシューマは、好ましい構成にアップグレードするように要求することができる。

【００１３】

一実施形態では、リソースマネージャは、一組のアプリケーションプログラムインタフェース（ＡＰＩ）を公開する。リソースコンシューマおよびリソースプロバイダは、ＡＰＩを使用して、リソースマネージャと通信し、リソースの登録、アクティビティの生成、構成の構築などの機能を実施する。

【００１４】

一実施形態では、リソースコンシューマは、リソースのうち、リソースコンシューマがタ

スクを実施するのに必要なリソースのサブセット（したがってそれらのリソースプロバイダ）だけしか意識しない。これらのリソースは、タスクを実施するためにリソースコンシューマの知らない他のリソースに依拠することもある。リソースプロバイダは、呼出しを受け取って構成を構築するように構成されている。リソースコンシューマが知っているリソースプロバイダは、リソースコンシューマから直接呼び出される。リソースコンシューマが知らないリソースプロバイダは、それらのリソースを使用するリソースプロバイダから呼び出される。

【００１５】

一実施形態では、リソースプロバイダが呼び出されたとき、リソースプロバイダは、リソースマネージャが１つ以上の構成を管理することを可能にする情報をリソースマネージャに提供する。ある特定の実装形態は、種々のリソースプロバイダ間のリソース従属関係を記述する階層ツリー構成である。この構成は階層的な性質を有するので、リソース予約およびリソースコンシューマへのエラー報告が容易である。

【００１６】

一実施形態では、リソース予約が失敗したとき、またはプリエンプト（ｐｒｅｅｍｐｔｉｏｎ）が行われたとき、エラー通知が生成される。この構成は階層的な性質を有するので、リソースコンシューマに知られているリソースプロバイダが見つかるまで各従属リソースプロバイダの親を辿っていくことにより、エラー報告はより効率的になる。次いで、この知られているリソースプロバイダは、リソースコンシューマに理解される用語で、エラーをリソースコンシューマに明確に伝えることができる。報告は様々な形をとることができる。例えば、報告は、要求された既知のリソースが利用不可能であることを知らせる単純な通知としてもよい。報告はまた、リソースコンシューマに種々のオプション（例えばタスクを実施するのに使用する代替リソース設定）を提示するものとしてもよい。

【００１７】

説明する実施形態の一態様は、リソースコンシューマにエラーを報告するのではなく、リソースプロバイダのレベルでエラーを修復することを試みる、トラブルシューティング機能を提供する。

【００１８】

一実施形態では、どのリソースが必要かをリソースコンシューマが知らなくてもいいようにリソースコンシューマのためにリソースマネージャとインタフェースするための、インテリジェントインタフェースコンポーネントが提供される。このインタフェースコンポーネントは、あるアクティビティにどのリソースが必要かを理解するように設計されている。インテリジェントインタフェースコンポーネントは、リソースコンシューマからの呼出しを受け取って特定の構成を構築することのできるプロキシリソースコンシューマとして行動する。この場合、インテリジェントインタフェースコンポーネントは、構成を構築してリソースの予約を要求するために、リソースマネージャと対話する。

【００１９】

一実施形態では、いわゆる「ステートレス」プロバイダを採用する。ステートレスプロバイダは、そのプロバイダが管理するリソースについてさえ、リソース割振りまたは所有権の情報を維持しないように設計されている。具体的には、かつ述べる実施形態では、ステートレスプロバイダは時間の概念、すなわち今要求されているのか将来要求されるのかについての概念を有さず、所与の要求があった時にどのリソースがどれだけ使用されているかについての概念しか有さない。別個のスケジューリングコンポーネントが、「もしも」シナリオを実行して、将来の選択された時にリソースが利用可能になるどうかを決定する。

【００２０】

各図面を通して、同じ要素および機構を参照するのには同じ番号を使用する。

【００２１】

（好ましい実施形態の簡単な説明）

本開示は、コンピュータシステム中のリソースを管理するためのリソース管理アーキテク

チャについて述べるものである。リソースは、様々なタスクまたは機能を実施するために利用される、コンピュータシステム中の限りある量のコンピューティングコンポーネントである。リソースの例には、ハードウェアデバイス、ポート、ＣＰＵ処理、メモリ、ＵＳＢ帯域幅、ネットワーク帯域幅、ソフトウェアモジュールなどが含まれる。リソースは、物理的ハードウェア量（例えばＣＰＵ、ＵＳＢ帯域幅、ネットワーク帯域幅）である場合もあり、抽象的な量（例えば仮想メモリ、オーディオ音量）である場合もある。

【００２２】

より多くのタスクを同時にかつ複数ユーザに対して実施することがコンピュータシステムに求められるのに伴い、限られたリソースの管理はますます重要になっている。例えば、単一のＴＶチューナを有するＴＶ対応コンピューティングシステム（例えばブロードキャストＰＣ、セットトップボックスなど）を考えてみる。処理を行うためにＴＶチューナを使用することの必要なプロセスが複数ある場合がある。例えば、ＴＶビューアアプリケーションは、モニタ上に表示するビデオストリームをＴＶチューナから提供してもらう必要がある。ＴＶレコーダアプリケーションもまた、後で再生する目的でエンコードしてハードディスクに録画するビデオストリームをＴＶチューナから提供してもらう必要がある。残念ながら、ＴＶチューナは一度に１つのＴＶチャネルにしか合わせることができない。システム中にＴＶチューナが１つしかない場合、システムは、ショーの視聴とショーの録画との両方を同時に行うことができないので（両方のアプリケーションが同じＴＶチャネルに合わせたいのでない限り）、これらの間で選択を迫られる。

【００２３】

別の状況では、おそらく複数のアプリケーションがユニバーサルシリアルバス（ＵＳＢ）上の帯域幅を同時に必要とする。あるアプリケーションは、存在するＵＳＢ帯域幅の２０％を消費する帯域幅要件を指定し、別のアプリケーションは１５％を消費する帯域幅を指定することがある。さらに、すべての要件が満たされるとすると、合わせた帯域幅が、利用可能なＵＳＢ帯域幅を超過することになると仮定する。このシナリオでは、１つ以上のアプリケーションは、ＵＳＢリソースへのアクセスを得ることができない恐れがあり、かつ／または要求した量よりも少ない量が割り振られる恐れがある。

【００２４】

本明細書で説明するリソース管理アーキテクチャの様々な実施形態は、特徴の中でもとりわけ、以下のような特性を有する。

【００２５】

・相対的な優先度に基づいて（最終的にはエンドユーザ定義のポリシーによって設定されたように）リソースを割り振る。
・リソースの割振りおよび割振り解除を動的に行う。
・優先度のより高いリソースコンシューマに再割当てすることを優先して、優先度のより低いコンシューマにリソースの返還を要求することができるようにする。
・追加のリソースが利用可能になったときにコンシューマに自動通知を提供する。
・リソースを動的に追加（登録）および除去（登録解除）できるようにする。
・リソース競合を詳細に報告するための機構を提供する。
・複数のリソースを自動的に獲得できるようにする。
・アクティビティの優先度を動的に変更できるようにする。
・リソース構成のランク付けを動的に変更できるようにする。
・リソースが公開されるように任意の第三者がリソースを登録することのできるフレームワークを提供する。
・コンシューマとリソースの任意の対がそれらに固有のプロパティを交渉することのできるリソース不可知フレームワークを提供する。
・リソース使用に実際に必要になる前にリソース割振りを行うことができるようにする（リソーススケジューリング）。
・コンシューマが最初にリソースを予約せずにリソースを使用することができないようにする機構を提供する。

・コンシューマがある範囲の構成を指定し、それらを最も望ましいものから最も望ましくないものまでランク付けすることができるようにする。
・プロセスがリソースを保持しながら異常終了したときにリソースを自動的に解放し、それによってリソースリークを防止するための機構を提供する。
・レガシーアプリケーションがリソース管理アーキテクチャと共存できるようにする。
【００２６】
このリソース管理アーキテクチャは、多くの様々な環境およびコンピューティングコンテキストで実現することができる。考察のために、このアーキテクチャは、ブロードキャスト対応パーソナルコンピュータ、セットトップボックス（ＳＴＢ）、ゲームコンソールなどの形をとる場合のある、消費者娯楽環境向けのコンピューティングシステムのコンテキストで述べる。リソースマネージャアーキテクチャを実現するのに適したシステムの１つについて述べた後、「一般的なリソース管理アーキテクチャ」の項目で、図２を参照しながらこのアーキテクチャについてより詳細に探求する。
【００２７】
（例示的なシステム）
図１に、コンピューティングユニット２２と、コンピューティングユニット２２と接続するか他の方法でインタフェースする複数の周辺コンポーネントとを有する娯楽コンピューティングシステム２０を示す。コンピューティングユニット２２は、１つ以上のプロセッサ２４（１）、２４（２）、．．．、２４（Ｐ）、揮発性メモリ２６（例えばＲＡＭ）、および不揮発性メモリ２８（例えばＲＯＭ、フラッシュ、ハードディスク、ＣＤ ＲＯＭなど）を有する。
【００２８】
不揮発性メモリ２８には、オペレーティングシステム３０が記憶されている。オペレーティングシステム３０はマルチタスキングオペレーティングシステムであり、揮発性メモリ２６にロードされて１つ以上のプロセッサ２４上で実行されるとき、複数のアプリケーション３２（１）、３２（２）、．．．、３２（Ａ）の同時実行をサポートする。好ましいオペレーティングシステムの１つは、本出願人から発売されているＷｉｎｄｏｗｓ（登録商標）ブランドのオペレーティングシステムである。ただし、他のオペレーティングシステムを採用してもよいことに留意されたい。
【００２９】
アプリケーション３２（１）－３２（Ａ）は、娯楽コンピューティングシステム２０上で実行することのできる多くの様々なタイプのアプリケーションプログラムを表す。例として、ＥＰＧ（電子番組ガイド）プログラム、ブラウザ、チャネルナビゲーション、オーディオビデオプレーヤ、オーディオ／ビデオ記録プログラム、ステレオプログラム、ゲーム、オーディオ／ビデオ遠隔会議などが含まれる。図では、ソフトウェアデコーダ３４（例えばＭＰＥＧソフトウェアデコーダ）およびその他のソフトウェアリソース３６も不揮発性メモリ２８に記憶されている。
【００３０】
オペレーティングシステム３０は、アプリケーション３２（１）－３２（Ａ）に割り振るために娯楽コンピューティングシステム２０のリソースを管理するリソース管理システム４０を有する。リソース管理システム４０は、オペレーティングシステム３０とは別に実装することもできるが、オペレーティングシステム内に統合されたものとして示す。リソース管理システム４０については、後で図２を参照しながらより詳細に述べる。
【００３１】
オペレーティングシステム３０はまた、コンピューティングシステム２０中の関連する様々な周辺コンポーネントのための複数のソフトウェアドライバ４２（１）、．．．、４２（Ｄ）も有する。１つ以上のＣＯＭ（通信）ポート４４もまた、オペレーティングシステム３０の一部として示されている。コンピューティングユニット２２を囲んで、周辺コンポーネントの代表的な集合が示されている。娯楽コンピューティングシステム２０は、テレビジョン番組などのブロードキャストデータを受信するための１つ以上の受信機５０を

有する。受信機５０は、アナログテレビジョン受信機、ディジタルテレビジョン受信機（例えば衛星放送受信アンテナやケーブルモデム）、ＲＦ受信機などとすることができる。受信機５０は、データを搬送する搬送波信号の周波数に同調する１つ以上のチューナ５２（１）－５２（Ｔ）に結合される。

【００３２】

コンピューティングユニット２２には、ＵＳＢバス５４が接続されて、多くの様々な種類のＵＳＢ適合周辺コンポーネントをインタフェースする。このようなコンポーネントの例には、モデム５６、スピーカ５８、スチルカメラまたはビデオカメラ６０、およびその他のＵＳＢデバイス６２が含まれる。

【００３３】

コンピューティングユニット２２には、１つ以上のハードウェアデコーダ６４（１）、６４（２）、．．．、６４（Ｈ）が結合されて、様々なタイプのデータストリームをデコードする。例示的なデコーダには、ＭＰＥＧ－１、ＭＰＥＧ－２、ＭＰＥＧ－４、Ｈ．２６１、Ｈ．２６３などの規格を用いるビデオデコーダや、オーディオデコーダが含まれる。

【００３４】

コンピューティングユニット２２は、ネットワーク６６に結合されて、他のコンピュータとインタフェースする。ネットワーク６６は、ＬＡＮ、ＷＡＮ、インターネット、イントラネット、無線ネットワークを含めた、多くの様々なタイプのネットワークを表す。リソース管理システム４０によって管理されるリソースの１つは、所与の時にネットワーク６６から提供される帯域幅である。

【００３５】

コンピューティングユニット２２には、１３９４シリアルバス６８が接続されて、多くの様々な種類の１３９４適合周辺コンポーネントをインタフェースする。このようなコンポーネントの例には、メモリドライブ７０（例えばディスクドライブ、テープドライブ、ＣＤ ＲＯＭドライブなど）、モデム７２、スピーカ７４、ＣＰＵ（中央処理装置）７６、およびその他の１３９４デバイス７８が含まれる。この例示的なシステム中にはＵＳＢバスおよび１３９４バスを示すが、ＳＣＳＩ、ＩＳＡ（Ｉｎｄｕｓｔｒｙ Ｓｔａｎｄａｒｄ Ａｒｃｈｉｔｅｃｔｕｒｅ）、ＰＣＩ（Ｐｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔ）バスなど、他のバスアーキテクチャを追加でまたは代替で使用することもできることに留意されたい。

【００３６】

娯楽コンピューティングシステム２０は、表示装置８０を有するが、これはテレビジョンセットまたはコンピュータモニタとすることができる。表示装置は、１つ以上の表示インタフェース８２（１）、８２（２）、．．．、８２（Ｃ）を介してコンピューティングユニット２２とインタフェースされる。これらの表示インタフェースは、ビデオポート、オーバーレイ、およびビデオメモリを表す。

【００３７】

コンピューティングデバイス２２に結合されるその他の例示的な周辺デバイスには、ＤＶＤプレーヤ８４、ＥＰＧデータベース８６、およびビデオレコーダ８８が含まれる。ＥＰＧデータベース８６は、ＥＰＧユーザインタフェース（ＵＩ）のタイルを埋める番組情報を保持する。番組情報は、番組タイトル、開始時間、継続時間、男優／女優、要約記述などの項目を含む。ＥＰＧ情報は、通常の手段を介して（例えばケーブルモデムを介して、または垂直帰線消去期間内に埋め込まれて）受信され、ＥＰＧデータベース８６に記憶される。コンピューティングユニット２２は、ＥＰＧデータベースに対して照会を実行して、ショーまたはその他の番組コンテンツを突き止め、この情報をグラフィカルＵＩ中でユーザに提示する。

【００３８】

ビデオレコーダ８８は、ビデオカセットレコーダやディスクベースのレコーダなどの形とすることができる。コンピューティングユニット２２は、チューナ５２またはネットワーク６６を介して受信された様々な番組を録画するようにビデオレコーダ８８に指示するこ

とができる。
【００３９】
さらに、娯楽に焦点を合わせた前述のコンポーネントに加えて、コンピューティングシステム２０は、コンピュータになじみの深い通常のデスクトップアプリケーションを実施できる完全に機能するコンピュータとして構成することができることにも留意されたい。システム上では、ワードプロセッシングアプリケーション、表計算アプリケーション、データベースアプリケーション、スケジューリングアプリケーション、財務アプリケーション、教育アプリケーションなど、様々なアプリケーションをロードして実行することができる。
【００４０】
図１に示すコンポーネントの集合は、リソース管理システム４０によって管理される例示的なリソースのタイプを示している。これらの中には、ポート、チューナ、デコーダ、ＵＳＢバス上のＵＳＢ帯域幅およびＵＳＢデバイス、ネットワーク帯域幅、１３９４デバイス、表示インタフェース、レコーダ、メモリドライブなどがある。その他の多くのコンポーネントをシステムに追加することもでき、図示のコンポーネントの１つ以上を除去することもできる。
【００４１】
（一般的なリソース管理アーキテクチャ）
図２に、図１の娯楽コンピューティングシステム２０によって実現される例示的なリソース管理アーキテクチャ１００を示す。アーキテクチャ１００はソフトウェア中で実現され、この例では、ユーザレベルのコンポーネントならびにカーネルレベルのコンポーネントを含む。
【００４２】
アーキテクチャ１００は、リソースマネージャ１０２および複数のプロバイダ１０４（１）、１０４（２）、１０４（３）、．．．、１０４（Ｐ）を有し、プロバイダは１つ以上のリソースコンシューマをサポートする。リソースコンシューマの例には、アプリケーション３２（１）、３２（２）、．．．、３２（Ａ）などユーザレベルのリソースコンシューマと、リソースコンシューマ３５などカーネルレベルのリソースコンシューマが含まれる。各プロバイダ１０４（１）－１０４（Ｐ）は、１つのリソースに関連し、リソースの可用性を追跡している。前述のようにリソースは、様々なタスクまたは機能を実施するのに利用される、コンピュータシステム中の限りある量のコンピューティングコンポーネントである。したがって、リソースプロバイダの例には、ハードウェアデバイスを所有するドライバ（例えばＴＶチューナのためのドライバ、バス上の帯域幅を所有するＵＳＢドライバ、ＣＰＵタイムリソースのためのＣＰＵスケジューラなど）、ハードウェアコンポーネント（例えばデコーダ）、およびソフトウェアモジュール（例えばソフトウェアデコーダ）が含まれる。さらに、単一のドライバが複数のリソースを提供する場合があることにも留意されたい。この場合、リソースマネージャ１０２はこのドライバを複数のプロバイダと見なす。リソースプロバイダをカーネルレベルに示すが、１つ以上のリソースプロバイダをユーザレベルに実装することもできる。
【００４３】
各プロバイダ１０４は、リソースマネージャ１０２による割振りに利用可能なリソースの量を決定するリソース定量化部（ｒｅｓｏｕｒｃｅ ｑｕａｎｔｉｆｉｅｒ）１０６を有する。リソース定量化部１０６は、所与のリソースの量がどのように測定されるかに応じて、種々の方式で可用性を計算するように構成される。ある方式は、有限のカウントを維持するものである。例えば、同調リソースのプロバイダに対するリソース定量化部１０６は、いくつのチューナが使用可能状態かを識別するカウンタとして実装することができる。
【００４４】
リソース可用性を計算するための別の方式は、パーセンテージとして計算するものである。ネットワークリソースのプロバイダに対するリソース定量化部１０６は、現在利用可能

な帯域幅のパーセンテージを計算するように実装することができる。時間ベースのメトリックを使用してリソース可用性を計算することもできる。このメトリックの例は、ＣＰＵが現在どれだけの処理時間を提供できるかを識別する、ＣＰＵに対するリソース定量化部である。所与のリソースの可用性を計算するための他の方式を用いることも、当然可能である。

【００４５】

各リソースプロバイダ１０４はリソースマネージャ１０２に登録し、リソースマネージャ１０２が情報を得るのに使用するコールバックのセットを供給する。例えば、あるコールバックはリソース計算を実施するのに使用され、別のコールバックは予約の成功をプロバイダに通知するのに使用される。

【００４６】

リソースマネージャ１０２は、リソースプロバイダ１０４から提供されるリソース（ローカルまたはリモート）へのアクセスを調停する。アプリケーション３２などのリソースコンシューマは、プロバイダ１０４から提供される１つ以上のリソースのセットを要求し、リソースマネージャ１０２は、どのアプリケーションがプロバイダのどのリソースを使用することになるかを決定する。リソースマネージャ１０２は、所定の競合解決機構に基づいてリソース割振り決定を行う。一実装形態では、競合解決機構は優先度ベースであり、したがってリソースマネージャ１０２は、優先度に基づいてリソースへのアクセスを調停する。別の実装形態では、競合解決機構は、所与の時に続行できるアクティビティの数を最大限にするように試みる負荷平衡に基づくものとすることができる。

【００４７】

別個かつ独立のポリシーマネージャ１０８を任意選択で実装して、リソースマネージャによって使用される競合解決機構に関するポリシーを設定することができる。例えば、リソースマネージャが優先度ベースの解決を採用している場合、ポリシーマネージャ１０８は、すべてのタスクにそれらのリソースを割り振ることができないような競合があるときに、どのタスクがリソースへのアクセスを得るべきかをリソースマネージャが決定できるように、ユーザまたはシステムによって与えられる相対的な重要度に従ってタスクをプリオリ（ｐｒｉｏｒｉ）にランク付けする。その他の実行可能なポリシーには、先着予約順、一番最近の予約順、リソースの「公平」共用、どれがどれに勝つかをユーザが選ぶポリシーなどが含まれる。多くの様々なポリシーが可能である。

【００４８】

システムまたはユーザがポリシー１１０を設定し、ポリシーマネージャ１０８がこれらを絶対的な優先度に変換する。リソースマネージャ１０８は、ユーザレベルとカーネルレベルの両方のコンポーネントによって実装することができる。

【００４９】

概して言えば、リソースコンシューマは、タスクを実施するためにリソースを必要とする任意のエンティティである。前述のように、アプリケーション３２はリソースコンシューマの一例である。別の例として、リソースプロバイダ自体が他のリソースのコンシューマである場合もある。考察のために、アプリケーションを１次コンシューマとし、したがってこの記述では、アプリケーション３２がリソースを要求および消費するものとして参照する。ただし、他のタイプのコンシューマを利用することもできるので、このことはこのアーキテクチャを限定するものと考えるべきではない。

【００５０】

リソースマネージャ１０２は、アーキテクチャ中の他のモジュールと対話するために、定義済みのＡＰＩ（アプリケーションプログラムインタフェース）１２０を公開する。ＡＰＩ１２０は、プロバイダ１０４によって使用されるプロバイダＡＰＩ呼出しのセットと、アプリケーション３２（１）－３２（Ａ）または他のリソースコンシューマからのリソース要求を受け入れるためのコンシューマＡＰＩ呼出しのセットとを含む。後の「リソースマネージャＡＰＩ」の項目で、ＡＰＩの１つについて詳細に述べる。

【００５１】

アプリケーション３２がタスクを実施したいとき、アプリケーション３２は、ＡＰＩ１２０を使用してリソースマネージャ１０２にアクティビティ１２２を生み出し、このアクティビティを実施するのに必要な様々なリソースセットを記述する１つ以上の構成１２４を構築する。アクティビティは、システム中で実施されるタスクに関連するデータ構造である。実施されるタスク１つにつき１つのアクティビティが存在する。図２では、リソースマネージャ１０２にアクティビティ１２２（１）、．．．、１２２（Ｎ）が示されている。リソースマネージャ１０２は、限られたリソースのプールからどのアクティビティを完全に満たすことができるかを決定し、満たすことのできるアクティビティを有するアプリケーションが要求のリソースにアクセスできるようにする。

【００５２】

構成は、システム中でタスクを実施するのに必要な対応するリソースに関する１つ以上のリソース記述子１２６の集合を保持するデータ構造である。アクティビティデータ構造は、１つ以上の構成を保持するコンテナである。図２では、第１のアクティビティ１２２（１）内に２つの構成１２４（１）および１２４（２）が示されている。第１の構成１２４（１）は、タスクを実施するのに必要なリソースを制御する対応するリソースプロバイダを識別するとともにこれらの必要なリソースの量を指定するための、４つの記述子R_1、R_2、R_3、R_4を含む。第２の構成１２４（２）は、タスクを実施するのに必要なリソースを制御する対応するリソースプロバイダを識別するとともにこれらの必要なリソースの量を指定するための、３つの記述子R_2、R_4、R_5を含む。アクティビティ１２２（１）は、構成１２４のいずれか一方を使用して、そのタスクを首尾よく完了することができる。

【００５３】

各リソース記述子１２６は、リソースコンシューマがタスクを実施するのに必要とするリソースのインスタンスを表す。リソース記述子は次の情報を含む。すなわち、（１）リソースを所有するリソースプロバイダ１０４の識別を保持するための識別フィールド１２８、（２）その構成に必要なリソースの量を保持するためのオプションの量フィールド１３０、および（３）１つ以上のリソース属性をリストするための属性フィールド１３２である。フィールド１３０に保持される量は、リソースマネージャには不明瞭であり、プロバイダおよびコンシューマだけに理解されればよい値である。同様に、属性も、プロバイダおよびコンシューマだけは理解するがリソースマネージャには不明瞭な、他のデータである。例えばチューナのコンテキストでは、リソース属性は、リソースコンシューマが欲するチューナ周波数とすることができる。

【００５４】

アクティビティ１２２（１）に対する２つの構成１２４（１）および１２４（２）で示すように、各アプリケーションは、各アクティビティにつき１つ以上の構成を指定する場合がある。構成は、アクティビティにリソースが予約されているかどうかに関わらず、いつでもアクティビティに追加することができる。複数の構成を指定する場合、アプリケーションは、好ましさまたは望ましさのレベルに従ってこれらをランク付けし、リソースマネージャ１０２は、最も望ましい構成を満たそうと試みる。ここで、第１の構成１２４（１）は、「Ｈ」で示すように、より好ましいか、望ましさがより高いものとして識別されている。第２の構成１２４（２）は、「Ｌ」で示すように、より好ましくないか、望ましさがより低いものとして識別されている。

【００５５】

複数の構成があることにより、リソースマネージャ１０２は、動作条件の変化に伴い、アプリケーションについてある構成から別の構成にフレキシブルかつ動的に変更することができる。例えば、リソースが優先度のより高いアプリケーションによってどこかで必要とされている場合、現在のアプリケーションに、より好ましくないまたは「フォールバック」構成を使用するように求めることができ、それにより、必要とされるリソースを優先度のより高いアプリケーションに再割振りすることができる。後でこれらのリソースが再び利用可能になったときは、リソースマネージャ１０２はアプリケーションに通知すること

ができ、したがってアプリケーションは、好ましい構成にアップグレードするように要求することができる。動的にフォールバック構成に変更し、より優先的な構成にアップグレードすることについては、後で「フォールバック構成」および「アップグレード通知」の項目でより詳細に述べる。

【００５６】

リソース記述子１２６は、ツリーに編成して、リソース間に内在する任意の依存を表すことができる。すなわち、リソースプロバイダもまた、他のプロバイダからのリソースを消費する。例えばＵＳＢカメラドライバは、カメラリソースのプロバイダであり、ＵＳＢバスドライバから提供される帯域幅のコンシューマである。このような関係を、リソース記述子のツリーとして表す。

【００５７】

ツリーの比喩では、構成１２４を記述子ツリーのルートと考えることができる。図２では、ツリーに編成された４つのリソース記述子１２６が第１の構成１２４（１）中にある。ツリーは、並んだ２つの兄弟ノードR_2およびR_4を含み、対応するリソースプロバイダから提供されるリソースが両方とも要求のタスクの実施に必要であることを表す。記述子R_4から子ノードR_3が分岐し、R_4で参照されるプロバイダが記述子R_3で参照されるリソースのコンシューマであることを示す。同様に、記述子R_3からノードR_1が分岐し、R_3で参照されるプロバイダが記述子R_1で参照されるリソースのコンシューマであることを示す。

【００５８】

（一般的な動作）

図３に、述べた実施形態によるリソース管理アーキテクチャ１００の一般的な動作を示す。このプロセスはソフトウェア中で実施されるものであり、このプロセスについて図２をさらに参照しながら述べる。

【００５９】

ステップ３００で、各リソースプロバイダ１０４（１）－１０４（Ｐ）は、それが管理するリソースをリソースマネージャ１０２に登録する。登録（および登録解除）はいつでも行うことができる。各リソースプロバイダ１０４は、リソースマネージャＡＰＩ１２０中の関数呼出し「ＲｍＲｅｇｉｓｔｅｒＲｅｓｏｕｒｃｅ」を使用して、そのリソースをリソースマネージャ１０２に登録する。各リソースには、リソースのタイプを識別するためのタイプ特有ＧＵＩＤ（Ｇｌｏｂａｌｌｙ Ｕｎｉｑｕｅ ＩＤｅｎｔｉｆｉｅｒ）が関連し、このＧＵＩＤが関数呼出しの一部として渡される。単一のプロバイダ１０４が複数のタイプを登録することができる。同様に、複数のプロバイダが同じリソースタイプに登録することもできる。

【００６０】

登録プロセスの一部として、リソースプロバイダ１０４は、リソースマネージャ１０２がコンシューマのためにリソースを予約および解放するのに使用することになるコールバックのセットを指定する。ＲｍＲｅｇｉｓｔｅｒＲｅｓｏｕｒｃｅプロセスは、リソースマネージャへの他の呼出しで使用されることになるハンドルを返す。

【００６１】

アーキテクチャ１００によれば、コンシューマは、リソースを予約および解放するためだけにリソースマネージャ１０２を経由する。他の場合は、コンシューマはリソースプロバイダ１０４に直接アクセスして、適切なリソース記述子を構成に追加し、リソースが予約されればそれらのリソースを使用する。これについては後でより詳細に述べる。

【００６２】

ステップ３０２で、リソースコンシューマは、１つ以上の関連するタスクを実施するために１つ以上のアクティビティを生み出す。例として、アプリケーション３２（１）がアクティビティ１２２（１）をリソースマネージャ１０２中に生み出すものとする。アプリケーション３２（１）は、関数呼出し「ＲｍＣｒｅａｔｅＡｃｔｉｖｉｔｙ」を使用してアクティビティをリソースマネージャ１０２に登録する。このプロセスは、アクティビティ

１２２（１）を形成するコンテナデータ構造を生み出し、アクティビティ１２２（１）についてのアクティビティハンドルを生成する。

【００６３】

ステップ３０４で、リソースコンシューマは、各アクティビティ内に１つ以上の構成を構築する。複数の構成を指定する場合、リソースコンシューマは、アクティビティ内の各構成を、このアクティビティの範囲内で望ましさからみてランク付けする。図２の例では、アプリケーション３２（１）は、関数呼出し「ＲｍＣｒｅａｔｅＣｏｎｆｉｇｕｒａｔｉｏｎ」を使用して、第１のアクティビティ１２２（１）内に少なくとも２つの構成１２４（１）および１２４（２）を構築する。このプロセスは、構成構造１２４（１）および１２４（２）へのハンドルを生成する。

【００６４】

構成１２４を生み出すと、リソースコンシューマは、関連するプロバイダを介してリソース記述子を構成に加えることを開始する（図３のステップ３０６）。例えば図２では、アプリケーション３２（１）は、対応するリソースプロバイダ１０４を使用してR_2やR_4などの記述子を第１の構成１２４（１）に加えることを開始する。プロバイダ１０４は、リソースマネージャＡＰＩからの関数呼出し「ＲｍＡｄｄＲｅｓｏｕｒｃｅＴｏＣｏｎｆｉｇｕｒａｔｉｏｎ」を利用して、記述子を構成に加える。記述子１２６を加えるとき、それらに対応するリソースプロバイダ１０４は、他のプロバイダ（通常はスタック中のより低いプロバイダ）を呼び出して、従属するリソース記述子を加えることができる。ここでは、記述子R_4で識別されるリソースプロバイダが、記述子R_3で識別されるリソースプロバイダを呼び出し、この記述子R_3で識別されるリソースプロバイダは、記述子R_1で識別されるリソースプロバイダを呼び出す。

【００６５】

ステップ３０８で、コンシューマはリソースマネージャと接触して、アクティビティのためのリソースを予約する。リソースマネージャは、リソースプロバイダと接触し、リソースが利用可能ならコンシューマのためにリソースを予約する。リソースマネージャは、記述子構成ツリー内で識別されるあらゆるリソースを予約することを試みる。リソースが利用可能でない場合、リソースマネージャはアクティビティ間で調停し、どのコンシューマにリソースへのアクセスを与えるかについての競合を解決する。コンシューマは、割り振られた量のリソースを、自発的に放棄するまで、またはリソースマネージャによってプリエンプトされるまで利用することができる。

【００６６】

予約および調停ステップ３０８の一実装形態を、サブステップ３２０－３３０に示す。ステップ３２０で、コンシューマが、アクティビティ１２２内のどの構成１２４を予約するかを指定する。構成を指定しない場合、リソースマネージャ１０２は、望ましい順に構成の予約を試みる。引き続き用いる例では、アプリケーション３２（１）は、関数呼出し「ＲｍＲｅｓｅｒｖｅＲｅｓｏｕｒｃｅｓ」を使用して、好ましい構成１２４（１）内で指定されるリソースを予約するようにリソースマネージャ１０２に指示する。この構成１２４（１）は、記述子R_1、R_2、R_3、R_4で識別されるプロバイダからのリソースを必要とする。

【００６７】

ステップ３２２で、リソースマネージャ１０２は、構成内の各リソース記述子１２６につき、対応するリソースプロバイダ１０４を識別し、システム中で現在そのリソースプロバイダからのリソースを使用しているアクティビティ１２２すべてのリストを作成する。予約することになるアクティビティ１２２（１）も、このリストに加える。リソースマネージャ１０２は、プロバイダから供給された「リソース割振り」関数を使用して、リストしたアクティビティ内のすべての記述子にリソースを割り当てる（ステップ３２４）。優先度ベースの手法では、リスト中のアクティビティには優先度が関連し、リソースマネージャ１０２は、リソースを割り当てる際、優先度の最も高いアクティビティから始める。リソースマネージャ１０２は、終わりに達するまで、またはプロバイダのリソースが尽きる

まで、アクティビティのリストの中を反復する。例示的なリソース計算についてのより詳細な記述を、次のセクションの「例示的なリソース予約計算」という項目で提供する。

【００６８】

構成内のリソース記述子がすべて予約されているとき、この構成は予約されているという。リソース記述子にリソースが割り当てられているとき、このリソース記述子は予約されているという。ステップ３２２および３２４は、構成内のあらゆるリソース／リソースプロバイダに対して繰り返す（ステップ３２６）。

【００６９】

ステップ３２８で、リソースマネージャ１０２は、すべての要求されたリソースが利用可能であるという点で各予約が成功したかどうかを決定する。予約が成功した場合は、リソースマネージャ１０２は、リソースプロバイダ１０４がコンシューマのリソース使用要求を妥当性検査することができるように、予約をリソースプロバイダ１０４に通知する（ステップ３３０）。このようにすることで、プロバイダ１０４は、最初に予約せずにリソースを使用しようとする不正なコンシューマならびに正当なコンシューマを捕えることができる。例えば、正当なプログラムが、ＣＰＵリソースを最初に予約せずに使用しようとする場合がある。同様に、ＵＳＢマウスが、ＵＳＢ帯域幅を最初に予約していないのに使用しようとする。このような状況で、ＣＰＵおよびＵＳＰ帯域幅に関連するプロバイダは、プログラムまたはＵＳＢマウスがまだリソースを予約していないと識別することができる。

【００７０】

予約が失敗した場合（要求されたリソースのすべてが利用可能ではないことを意味する）、リソースマネージャ１０２は、予約失敗をリソースプロバイダ１０４に通知する（ステップ３３２）。次いでリソースマネージャ１０２は、次のタスクの一方または両方を実施する。すなわち、（１）より好ましくない構成が利用可能であって、予約すべき構成をコンシューマが指定していなかった場合は、より好ましくない構成を試みる（ステップ３３４）、かつ／または（２）要求元のコンシューマにエラーを報告する。

【００７１】

（例示的なリソース予約計算）

図４に、図３の予約および調停ステップ３０８（より具体的にはステップ３２２－３２６）の一部として採用される例示的なリソース予約プロセスを示す。図３のステップ３０８で、リソースマネージャ１０２は、アプリケーション３２などのコンシューマのためにリソースを予約する。図４には、３つのアクティビティＡ１、Ａ２、Ａ３が示してあり、これらを番号４００（１）、４００（２）、４００（３）で一般的に参照する。これらのアクティビティは、リソースマネージャ１０２で構築され、リソースマネージャ１０２にある。各アクティビティには優先度が関連し、したがって、第１のアクティビティＡ１は最高優先度を有し、第２のアクティビティＡ２は中間優先度を有し、第３のアクティビティＡ３は最低優先度を有する（すなわちＡ１＞Ａ２＞Ａ３）。

【００７２】

アクティビティの「優先度」は、ユーザまたはシステムの好ましさを示す。これは、従来のスレッド優先度またはプロセス優先度の意味で用いるものではない。具体的には、アクティビティ優先度は、同じリソースを求めて争うコンシューマ間の調停ポリシーを規定するものである。述べた実装形態では、ポリシーマネージャ１０８が各アクティビティに優先度を割り当てる。優先度は、個々のリソースとは対照的に、アクティビティに割り当てることに留意されたい。このようにすることで、リソースマネージャは、優先度のより低いアクティビティにリソースの返還を要求して、優先度のより高いアクティビティの予約要求を満たすことができる。

【００７３】

この例では、各アクティビティ４００（１）－４００（３）は、ただ１つの構成４０２（１）－４０２（３）を有する。記述子R_1およびR_2は、リソースマネージャに登録している対応するリソースプロバイダを表す。アクティビティＡ１およびＡ２は両方とも記述

子R_1およびR_2を有するが、アクティビティＡ３は記述子R_1しか有さない。この例で、最高優先度のアクティビティおよび最低優先度のアクティビティＡ１およびＡ３が予約されており、リソースマネージャが、中間優先度のアクティビティＡ２を予約する要求を受けると仮定する。リソースマネージャは、以下の５つのステップを実施してＡ２を予約する。

【００７４】

ステップ１：リソースマネージャは、その内部状態を調べ、予約されているすべてのアクティビティのアクティビティリストを作成する（すなわち図３のステップ３２２）。この時点では、リストは予約済みのアクティビティＡ１およびＡ３を含む。次いでリソースマネージャは、アクティビティＡ２をリストに加え、得られたリストを優先度の高い順にソートする。この結果、最高優先度のアクティビティＡ１、次いで中間優先度のアクティビティＡ２、次いで最低優先度のアクティビティＡ３を順に含むアクティビティリストが得られる。

【００７５】

図５に、３つのアクティビティを優先度の順に表すアクティビティリスト５００を示す。アクティビティリスト５００はリソースマネージャによって維持される。

【００７６】

ステップ２：アクティビティＡ２内の各リソース記述子につき、リソースマネージャは、要求を満たすのに十分なリソースがあり、したがってリソースマネージャがそのリソース記述子をこのアクティビティのために予約することが可能であるかどうかを決定する。これはまず、アクティビティリスト５００中のすべてのアクティビティにおける、予約すべきリソース記述子で参照されるプロバイダと同じプロバイダを使用するリソース記述子のリストを作成することによって行う。例えば、Ａ２－R_1（すなわちアプリケーションＡ２内の記述子R_1）を予約するために、リソースマネージャは、アクティビティリスト５００中にリストされているアクティビティ内のすべてのリソース記述子R_1に関する別個のリソース記述子リストを構築する。このリストもまた、優先度の高い順である。

【００７７】

図５に、２つの記述子リスト５０２（１）および５０２（２）を示す。第１のリスト５０２（１）は、アクティビティリスト５００中にリストされているアクティビティＡ１－Ａ３内のすべてのリソース記述子R_1を、優先度の高い順に含む。したがって、Ａ１－R_1－１、Ａ１－R_1－２（すなわちアクティビティＡ１内におけるプロバイダR_1の第２の使用）、Ａ２－R_1、Ａ３－R_1の順である。第２のリスト５０２（２）は、アクティビティリスト５００中にリストされているアクティビティＡ１－Ａ３内のすべてのリソース記述子R_2を、優先度の高い順に含む。したがって、Ａ１－R_2、Ａ２－R_2－１、Ａ２－R_2－２の順である。記述子リスト５０２は、リソースマネージャによって維持される。

【００７８】

ステップ３：記述子リスト５０２が完成した後、リソースマネージャは、様々なアクティビティに割り振られるリソースの量の継続的な増加を表す累積値５０６を保持するバッファ５０４を生み出す。リソースマネージャは、R_1記述子リスト５０２（１）中の各要素につき、リソースプロバイダR_1の「アキュムレータ追加」関数を呼び出し、リソース記述子（すなわちＡ１－R_1－１、Ａ１－R_1－２など）、アキュムレータバッファ５０４、およびリソースプロバイダのリソース定量化部１０６（図２）を渡す。リソースマネージャは、R_2記述子リスト５０２（２）中の各要素についても同様の呼出しを行う。

【００７９】

アキュムレータ追加関数は、リソース記述子に必要なリソース量を決定し、それをアキュムレータバッファ５０４の内容に加える。バッファ５０４中の新しい値がリソースプロバイダR_1におけるリソースの最大量を超える場合は、アキュムレータ追加関数はエラーを返し、プロバイダがリソース不足によりこの割振りを満たすことができないことを示す。リソースマネージャは、このようなリソース記述子に関連するアクティビティを「犠牲者

」としてタグ付けする。例えば、リソース記述子Ａ３－Ｒ$_1$に対する計算が失敗した場合、アクティビティＡ３は犠牲者としてタグ付けされる。予約されるアクティビティＡ２が犠牲者としてマークされた場合は、リソースマネージャは図３のステップ３３２に出て、アクティビティＡ２内の構成を予約することが不可能であることを通知する。

【００８０】

ステップ４：リソースマネージャは、ステップ２で構築したリソース記述子リスト中のすべてのリソース記述子を処理した後、犠牲者アクティビティがあるかどうか評価する。犠牲者アクティビティがない場合、リソースマネージャはアクティビティＡ２の予約を成功させることができた。Ａ２内のすべてのリソース記述子のプロバイダに、新しい予約が通知される。これによりプロバイダは、コンシューマのリソースアクセス要求を妥当性検査することができ、また逆に、予約なしでリソースを使用しようとする不正コンシューマを捕えることができる。

【００８１】

ステップ５：一方、ステップ３の最後で犠牲者アクティビティがある場合は、リソースマネージャは、リソースを解放するようにこれらのアクティビティに通知する。リソースが解放されると、リソースマネージャは、元々要求を出したアクティビティ（この場合はＡ２）にこれらのリソースを割り当てる。例えば、ステップ３でアクティビティＡ３が犠牲者としてタグ付けされたとする。リソースマネージャは、アクティビティＡ３にそのリソースを解放するように通知し、それらを要求元のアクティビティＡ２に再割振りする。これは、アクティビティＡ２がアクティビティＡ３よりも高い優先度を有し、したがって限られたリソースを優先度のより低いアクティビティＡ３からより高いアクティビティＡ２に移すべきなので、理にかなっている。

【００８２】

（優先度ベースのプリエンプトを用いたリソース割振り）

図２に示すリソース管理アーキテクチャ１００は、種々のタイプの戦略を採用して、リソースコンシューマから要求される様々なアクティビティ間でリソースを割り振ることができる。例えばある戦略は、できるだけ多くのアクティビティ間でリソースを最適化するものである。別の戦略は、何らかの優先度基準に従ってリソースを割り振るものである。この後者の戦略が、このセクションにおける関心の対象である。

【００８３】

優先度ベースの戦略では、どのアプリケーションおよび／またはユーザが他を制してリソース使用の優先度を有するかに基づいてリソースを割り振る。繰り返すが、用語「優先度」は、従来のスレッド優先度やプロセス優先度の意味では用いず、同じリソースを求めて争うコンシューマ間の調停ポリシーのコンテキストで用いる。優先度は、個々のリソースとは対照的に、アクティビティに割り当てる。

【００８４】

優先度ベースの戦略では、アクティビティを重要度によってランク付けすることができる。例えば、ユーザにとってより「重要な」アクティビティを、より高い優先度に指定することができる。これによってリソース管理アーキテクチャは、所望の、ただし限られたリソースを、より重要でないアクティビティから取り去ってより重要なアクティビティに移行することができる。何が「より重要な」ものおよび「より重要でない」ものを構成するかを決定することが、ポリシーマネージャ１０８およびポリシー１１０の場であり、これについては後で「ポリシーマネージャ」の項目でより詳細に述べる。しかし、プリエンプトに関するここでの考察では、他のアクティビティに対する相対的な重要度など、何らかの方式でアクティビティをランク付けする何らかの優先度等級があるものとする。例として、ポリシーマネージャ１０８がない場合、アクティビティ優先度は先着順で割り当てることができる。

【００８５】

優先度ベースのプリエンプトにより、リソースマネージャは効果的に、リソースを現在使用している優先度のより低いアクティビティを「プリエンプト」し、優先度のより高いア

クティビティにリソースを動的に移動する。リソースマネージャは、優先度のより低いアクティビティに、そのリソース使用権が一時停止されることを通知する。これにより、この優先度のより低いアクティビティには、制御されたクリーンな方式でリソースの使用を停止する機会が与えられるか、または、これに反応して、場合によっては代替方式でその処理を完了するチャンスが与えられる。

【００８６】

リソースマネージャは、リソースの返還を要求する対象のアプリケーションと連携して、プリエンプトプロセスを達成する。図２に示すように、リソースマネージャ１０２は、優先度が割り当てられた１つ以上のアクティビティ構造１２２をシステム内で維持する。各アクティビティは１つ以上の構成構造１２４を有し、構成構造１２４は、アクティビティがそのタスクを達成するのに必要なリソースのセットを指定する。リソースマネージャ１０２はまた、どのアクティビティにその構成のためのリソースが与えられているかを追跡し、各リソースプロバイダ１０４との必要な通信と、各リソースプロバイダに関する必要な状態とを維持して、各リソースプロバイダの提供するリソースが現在どれだけ利用可能かを追跡する。

【００８７】

図６に、優先度ベースのプリエンプトを用いたリソース割振りプロセスを示す。各ステップは、リソースマネージャおよびリソースプロバイダによって（図示のように）、ソフトウェア中で実施されるものであり、これらのステップについて図２をさらに参照しながら述べる。

【００８８】

ステップ６００で、リソースマネージャ１０２は、あるタスクを実施するためのアクティビティ１２２を生み出すことを求める要求をコンシューマ（例えばアプリケーション）から受け取る。アクティビティの一部として、１つ以上の構成１２４が指定される。次いで、コンシューマは構成を予約する。リソースマネージャ１０２は、構成１２４内で識別される登録済みの各リソースプロバイダ１０４に、そのリソースをアクティビティ１２２に割り振ることができるかどうか尋ねる（ステップ６０２）。

【００８９】

新しい構成を満たすのに十分なリソースをプロバイダ１０４が有する場合、リソースマネージャは、リソースをアクティビティに割り振る（すなわちステップ６０４からの「ｙｅｓ」ブランチおよびステップ６０６）。そうではなく、いずれかのプロバイダ１０４に構成を満たすのに十分なリソースが残っていない場合は、このプロバイダ１０４は、リソースの不足をリソースマネージャ１０２に通知する（すなわちステップ６０４からの「ｎｏ」ブランチおよびステップ６０８）。

【００９０】

リソースマネージャ１０２は、リソースを現在要求しているアクティビティよりも優先度の低いすべてのアクティビティ１２２のすべての構成１２４をチェックして、優先度のより低いアクティビティのいずれかが、優先度のより高い新しいアクティビティに割り振ればその構成を満たすことになるリソースを現在使用しているかどうかを決定する（ステップ６１０）。このような優先度のより低いアクティビティまたはその組合せが存在しない場合は（すなわちステップ６１２からの「ｎｏ」ブランチ）、リソースマネージャ１０２は、優先度のより高いアクティビティに、そのリソース構成を現在満たすことができないことを通知する（ステップ６１４）。

【００９１】

一方、リソースを取り去ることのできる優先度のより低いアクティビティが存在する場合は（すなわちステップ６１２からの「ｙｅｓ」ブランチ）、リソースマネージャ１０２は、この優先度のより低いアクティビティ内で現在予約されているリソースを含めて、優先度のより高い新しいアクティビティ内のリソースすべてを予約できるかどうかを決定する（ステップ６１６）。予約できない場合は（すなわちステップ６１６からの「ｎｏ」ブランチ）、リソースマネージャ１０２は、優先度のより高いアクティビティに、そのリソー

ス構成を現在満たすことができないことを通知する（ステップ６１４）。反対に、リソースを予約できる場合は、リソースマネージャ１０２は、優先度のより低い各アクティビティに、そのリソースを放棄するように知らせるプリエンプト通知を送る（ステップ６１８）。この場合、優先度のより低いアクティビティには、制御された方式でそのリソース予約を減らして、優先度のより高いアクティビティに再割振りされるようにリソースを解放する機会が与えられるか、または、これに反応して、場合によっては代替方式でその処理を完了するチャンスが与えられる（ステップ６２０）。優先度のより低いすべてのアクティビティが現在使用しているリソースをすべて解放したとき、リソースマネージャ１０２は、優先度のより高い新しいアクティビティに、その必要とするリソースの割振りに進むことができることを通知する（ステップ６２２）。このようにして、優先度のより低いプロセスは、優先度のより高いリソースによって必要とされるその限られたリソースの使用をプリエンプトされる。

【００９２】

優先度のより低いプロセスが、リソースマネージャからプリエンプト通知を受け取ったときにそのリソースを快く放棄しない場合は、リソースマネージャは、そのアクティビティの予約を取り消して、かつ／または優先度のより低いアクティビティに関連するプロセスを終了して、強制的にそのリソースの返還を要求することができることに留意されたい。

【００９３】

図７－１０に、優先度ベースのプリエンプトを用いたリソース割振りを例示するための種々のシナリオを示す。各図は、リソースマネージャ１０２に存在する１つ以上のアクティビティと、アクティビティをもう１つ追加する試みとを示す。既存のアクティビティおよび新しいアクティビティにはそれぞれ、優先度が関連する。考察しやすいように、各アクティビティは構成を１つだけ有するものとする。

【００９４】

図７には、２つのアクティビティが同じリソースを利用したいと思っているが、残念ながら２つのアクティビティのうちの一方にしかリソースを割り振ることができない場合を示す。例えば、リソースがチューナであり、システムがチューナを１つしか有さないと仮定する。２つのアクティビティＡ１およびＡ２（番号７００（１）および７００（２）で一般的に参照する）は、対応する構成７０２（１）－７０２（２）を有し、各構成は、対応する同じリソースプロバイダを表す同一の記述子を含む。

【００９５】

図７の場合、既存のアクティビティＡ１は最高優先度を有し、追加が求められる新しいアクティビティＡ２は最低優先度を有する（すなわちＡ１＞Ａ２）。図６の優先度ベースのプリエンプトプロセスに従って、リソースマネージャは、まず記述子R_1で識別されるリソースプロバイダと接触することにより、Ａ１に対して「リソース計算」を実行する。先に「リソース計算」方法の中で述べたように、プロバイダは、この記述子によって必要とされるリソース量をアキュムレータに加える。アキュムレータの新しい値は１になる。プロバイダは、アキュムレータの値を、それが有するリソースの総量（この場合は１）と比較し、この量を満たすことができることをリソースマネージャに示す。リソースマネージャは、「アキュムレータ」を１に初期化した状態で、アクティビティＡ２に対してこれと同じ手順を繰り返す。プロバイダは、アクティビティＡ２に必要なリソースの量をアキュムレータの内容に加え（図６のステップ６０２）、これがプロバイダの有するリソースの総量を超えることに気付く。リソースプロバイダは、現在の割振りでは要求を満たすことができない旨の通知を返す（ステップ６０４および６０８）。次いでリソースマネージャは、要求されるリソースを現在使用している優先度のより低いアクティビティがあるかどうか評価する（ステップ６１０）。この場合、このリソースの現ユーザは既存のアクティビティＡ１であり、アクティビティＡ１は、新しいアクティビティＡ２よりも優先度が高い。したがってリソースマネージャは、新しいアクティビティに、今回はその構成を満たすことができないことを通知する（ステップ６１２および６１４）。

【００９６】

図８に、図７と同様だが、例外として新しいアクティビティの方が既存のアクティビティよりも高い優先度を有する場合を示す。２つのアクティビティＡ１およびＡ２（番号８００（１）および８００（２）で一般的に参照する）は、対応する構成８０２（１）－８０２（２）を有し、各構成は、対応するリソースプロバイダを表す記述子を含む。この場合、既存のアクティビティＡ１の方が新しいアクティビティＡ２よりも低い優先度を有する（すなわちＡ１＜Ａ２）。

【００９７】

図６の優先度ベースのプリエンプトプロセスに従って、リソースマネージャは、記述子R_1で識別されるリソースプロバイダに尋ねて、新しいアクティビティＡ２にリソースを割り振ることができるかどうかを決定する（図６のステップ６０２）。リソースプロバイダは、現在の割振りでは要求を満たすことができない旨の通知を返す（ステップ６０４および６０８）。

【００９８】

次いでリソースマネージャは、要求されるリソースを現在使用している優先度のより低いアクティビティがあるかどうか評価する（ステップ６１０）。この場合、このリソースの現ユーザは既存のアクティビティＡ１であり、アクティビティＡ１は、新しいアクティビティＡ２よりも優先度が低い。したがってリソースマネージャは、優先度のより低いアクティビティＡ１にプリエンプト通知を送り（ステップ６１６）、優先度のより低いアクティビティが停止または完了できるようにする（ステップ６１８）。リソースマネージャは、優先度のより高い新しいアクティビティＡ２に、その要求するリソースが割り振られることを知らせる（ステップ６２０）。したがって、図８は、リソースマネージャが既存のアクティビティから新しいアクティビティに動的に移行し、それによって優先度のより高い新しいアクティビティのために既存のアクティビティを犠牲にする場合を表す。

【００９９】

図９に、犠牲者アクティビティがそのリソース使用を解放するか減らすように求められる場合を示す。この場合、犠牲者アクティビティは現在、他のリソースR_1、R_2、R_4も予約しており、これらのリソースはリソースマネージャから要求されていない。このような場合、アクティビティを所有するコンシューマは、すべてのリソースを解放するか、リソースマネージャから要求されるリソースのすべてを解放するか、リソースマネージャから要求されるリソースの一部を開放するかを決定することができる。考察のために、リソースR_3がＣＰＵリソースに関係し、アクティビティＡ１に関連するコンシューマが、ＣＰＵの５０％を求めるアクティビティＡ２からの要求に応答して、そのＣＰＵ使用を４０％使用に減らしたいと思っていると仮定する。

【０１００】

図９では、既存の低優先度のアクティビティＡ１（番号９００（１）として一般的に参照する）が、対応する構成９０２（１）を有し、この構成は、記述子R_1－R_4で表される４つのリソースを含む。新しい高優先度のアクティビティＡ２（番号９００（２）として一般的に参照する）が、対応する構成９０２（２）を有し、この構成は１つのリソースR_3しか含まない。リソースマネージャは、高優先度のアクティビティＡ２が、記述子R_3で識別されるリソースに対する権利を有すると決定し、アクティビティＡ１をプリエンプトする。この場合、犠牲者アクティビティＡ１は、記述子R_3で識別されるリソースの使用を減らすことを選択し、記述子R_1、R_2、R_4で識別される残りのリソースの制御は、少なくとも当面は維持する。

【０１０１】

図１０に、リソースマネージャが、既存の低優先度のアクティビティをプリエンプトする行程に乗り出す前に、新しい構成の構成全体を満たすことができるかどうかを考慮する場合を示す。図１０には、既存の低優先度のアクティビティＡ１（すなわち１０００（１））を示すが、このアクティビティＡ１は、単一の記述子R_1をリストする構成１００２（１）を有する。別の既存のアクティビティだが高優先度のアクティビティＡ２（すなわち１０００（２））は、単一の記述子R_2をリストする構成１００２（２）を有する。中間

優先度の第３のアクティビティＡ３（すなわち１０００（３））が、追加されるように要求する。第３のアクティビティＡ３は、２つの記述子R_1およびR_2をリストする構成１００２（３）を有する。

【０１０２】

この場合、リソースマネージャは、新しいアクティビティＡ３の要求を満たすために、２つの記述子R_1およびR_2で識別されるリソースを割り振らなければならない。リソースマネージャは、低優先度のアクティビティＡ１をプリエンプトして、記述子R_1で識別されるリソースを中間優先度のアクティビティＡ３に再割振りするのを可能にすることもできるが、これは１つのリソースを満たすだけである。リソースマネージャはさらに、他方のリソースR_2の現ユーザが高優先度のアクティビティＡ２であると決定する。したがってリソースマネージャは、中間優先度のアクティビティＡ３のために高優先度のアクティビティＡ２をプリエンプトすることはしない。リソースマネージャは新しいアクティビティＡ３の構成全体を満たすことはできないので、低優先度のアクティビティＡ１をプリエンプトしないことを選択し、その代わり、新しいアクティビティＡ３に関連するコンシューマに、その要求を満たすことができないことを通知する。

【０１０３】

（フォールバック構成）

前のセクションで述べた各シナリオでは、アクティビティが構成を１つだけ有すると仮定した。しかし、図２の一般的なリソース管理アーキテクチャに示すように、各アクティビティが複数の構成を指定することもある。複数の構成がある場合、アクティビティがリソース使用をプリエンプトされてもなお、プリエンプトされたリソースを必要としない別の構成を使用して処理を継続することができる状況があり得る。したがって、フレキシブルな適合プロセスが、代替リソースセットを使用して、何らかの代替方式でそのタスクを完了することができる場合がある。例えば、アクティビティは、ハードウェアビデオデコーダ（優先度のより高いアクティビティに再割振りされるリソース）を使用する代わりに、ソフトウェアビデオデコーダを使用するアルゴリズムを別法として採用することができる。

【０１０４】

代替構成は、競合しないであろう異なるリソース、または単により少ないリソースを利用するものとすることができる。通常、代替構成を使用することは、アプリケーションによって生み出される結果の品質を落とすことにつながる。この状況では、アクティビティがより好ましい構成からより好ましくない別の構成に「フォールバック（後退）」するという。例えば、アクティビティが、ハードウェアビデオデコーダを有する好ましい構成からソフトウェアビデオデコーダを有するフォールバック構成に動的に変更するとき、フォールバック構成は、全画面サイズの画像をデコードすることができない。したがって、アプリケーションの出力の品質が落ち、モニタ上のビデオ画像のサイズが縮小する。

【０１０５】

図２を参照するが、リソースマネージャアーキテクチャ１００は、リソース割振りが変更されたときにアクティビティ１２２がフレキシブルに適合することを容易にする。各アクティビティ１２２は、複数の構成１２４（１）、１２４（２）などを含むことができ、各構成は、プロセスが構成内のすべてのリソースを有する場合にプロセスが進行できるような、許容されるリソースセットを表す。構成１２４は、アプリケーションまたはコンシューマプロセスによって好ましさの順にランク付けされる。リソースマネージャ１０２がアクティビティをプリエンプトするとき、アクティビティは、そのリソース使用を解放するか減らす必要がある。リソースが解放された後は、いくつかの動作進行が考えられる。ある手法では、プリエンプトされたアクティビティに関連するコンシューマは、別の構成のリソースを予約するようにリソースマネージャに頼むことができる。別の手法では、リソースマネージャ１０２は自動的に、次に高いランクのフォールバック構成内のリソースを予約することを試みる。リソースマネージャは、（１）現在利用可能なリソースで満たすことのできる構成を見つけるまで、または（２）満たすことのできるフォールバック構成

がないとわかるまで、各フォールバック構成に連続して進む。
【０１０６】
図１１に、動作条件の変化に伴ってより好ましい構成からより好ましくない構成に動的にダウングレードするプロセスを示す。このような変更は、例えば、リソースマネージャ１０２が高優先度のアクティビティにリソースを再割振りするために低優先度のアクティビティをプリエンプトする、優先度ベースのポリシーの結果として引き起こされることがある。図１１の各ステップは、リソースマネージャ１０２によって実施されるものであり、これらのステップについて図２をさらに参照しながら述べる。図示のプロセスでは、リソースマネージャ１０２が、例えば図６に示したリソース割振りプロセスの結果として、アクティビティによる１つ以上のリソースの使用をプリエンプトしたと仮定する。このため、犠牲者アクティビティには、もはやその好ましい構成が利用可能ではない。
【０１０７】
ステップ１１００で、リソースマネージャ１０２がアクティビティをプリエンプトしたとき、リソースマネージャ１０２は、犠牲者アクティビティ１２２が別の構成１２４を有するかどうかを決定する。アクティビティコンテナ１２２内で他の構成が指定されていない場合は（すなわちステップ１１００からの「ｎｏ」ブランチ）、リソースマネージャは、代替構成が指定されていないことをコンシューマ（例えばアプリケーション３２）に通知し、フォールバックリソースセットを記述する新しい構成をアクティビティ内に生み出してこの新しい構成の予約を要求する機会をコンシューマに与える（ステップ１１０２）。構成の生成については、図３に関して先に述べてある。
【０１０８】
あるいは、１つ以上の他の構成がアクティビティ内に示されている場合は（すなわちステップ１１００からの「ｙｅｓ」ブランチ）、リソースマネージャ１０２は、アクティビティ１２２内のすべてのフォールバック構成１２４の中を進む（ステップ１１０４）。リソースマネージャ１０２は、満たすことのできるフォールバック構成があるかどうか決定する（ステップ１１０６）。満たすことのできる構成がない場合は（すなわちステップ１１０６からの「ｎｏ」ブランチ）、リソースマネージャはコンシューマに通知し、新しい構成をアクティビティ内に生み出す機会をコンシューマに与える（ステップ１１０２）。
【０１０９】
反対に、少なくとも１つのフォールバック構成を満たすことができる場合は（すなわちステップ１１０６からの「ｙｅｓ」ブランチ）、リソースマネージャ１０２はこの構成内のリソースを予約する（ステップ１１０８）。リソースマネージャ１０２はまた、フォールバック構成を満たすことができることをコンシューマに通知する（ステップ１１１０）。通知は、プリエンプトされていないリソースが現在の構成とフォールバック構成の両方にある場合に、またはプリエンプトされたリソースが現在の構成とフォールバック構成の両方にあって、フォールバック構成の方が現在の構成よりも少ない量のプリエンプトリソースを使用する場合に、プロセスがそのリソースを解放せずに構成を切り替えることができるような形で行う。
【０１１０】
図１１のフォールバックプロセスは、異なる３つの手法をサポートする。ある手法では、コンシューマの現在の構成がプリエンプトされることがコンシューマに通知されるまで、代替構成を生み出さなくてもよい。これは、図１１のステップ１１０２によって例示するものである。この時点でコンシューマ（例えばアプリケーション３２）は、競合リソースを含まない新しい構成を生み出し、リソースマネージャ１０２に、代替構成をアクティビティに加えて代替構成を予約するように伝える。代替構成の構築にはいくらか時間がかかることがあり、リソースマネージャとの通信（およびプロバイダとの間接的な通信）にも時間がかかることがある。この時間遅延は品質に影響を及ぼすことがある。
【０１１１】
フォールバック構成内のすべてのリソースが利用可能な場合、リソースマネージャ１０２は、フォールバックリソースを予約し、コンシューマが望むなら元のリソースを解放する

。あるいは、コンシューマは、２つの構成の結合を一時的に保有し、後で元のリソースを明示的に解放することもできる。フォールバック構成を満たすことができない場合は、元の構成内のリソースは解放されず、したがって、プロセスは別のフォールバック構成を用いて再び試みる。最終的に、許容されるフォールバック構成を生み出すことができない場合は、コンシューマは処理を停止し、リソースを別のアクティビティに割り振られるように解放する。次いでコンシューマは、リソースが再び利用可能になるまで待機する。

【０１１２】

この第１の手法の変形は、コンシューマが、その通知を受け取ったときに、プリエンプトされたリソースを解放できることをリソースマネージャに明示的に通知するものである。このようにすると、コンシューマがフォールバック構成を構築できるまで待機するよりも少し早くリソースを解放することができる。

【０１１３】

別の手法では、代替構成はリソースコンシューマによって事前に生み出される。これは、図１１のステップ１１００からの「ｙｅｓ」ブランチと、図２のアクティビティの１つ１２２（１）内の複数の構成１２４（１）および１２４（２）によって示すものである。プリエンプトの一部として、リソースマネージャ１０２は単に、プリエンプトされるコンシューマに、アクティビティ内のその現在構成を失うことになり所与のフォールバック構成に切り替えなければならないことを通知する。リソースマネージャは、この通知の前にフォールバック構成内のリソースを予約し、したがって、コンシューマは一時的に２つの構成の結合を保有する。コンシューマは次いで競合するリソースを解放し、代替のリソース集合をサポートするための必要な変更を行う。次いでリソースマネージャ１０２は、元の構成内にあってフォールバック構成内にはない争われたリソースを解放し、それによってこれらのリソースが他のアクティビティに割り振られるようにする。

【０１１４】

フォールバック構成はまた、最初にリソースを予約する間に使用することもできることに留意されたい。例えば、アプリケーション３２が価値でランク付けした複数の構成１２４をアクティビティ１２２内に供給するとき、リソースマネージャ１０２は、リソースマネージャ１０２が満たすことのできる最もランクの高い構成を選び、これらのリソースを予約し、どの構成が満たされたかをプロセスに通知する。

【０１１５】

図１２に、より好ましくない構成にダウングレードするプロセスにおけるシナリオの１つを示す。図１２には、リソースマネージャ１０２にある、既存のアクティビティＡ１（番号１２００（１）として参照する）および新しいアクティビティＡ２（番号１２００（２）として参照する）を示す。新しいアクティビティＡ２は、既存のアクティビティＡ１よりも優先度が高い。

【０１１６】

既存のアクティビティＡ１は、２つの構成を有する。すなわち、好ましい構成Ｃ１（番号１２０２として参照し、より高くランクされていることを示すための文字「Ｈ」を付けてある）と、フォールバック構成Ｃ２（番号１２０４として参照し、より低くランクされていることを示すための文字「Ｌ」を付けてある）である。好ましい構成Ｃ１は、アクティビティを実施するのに必要な対応するリソースを識別するためのリソース記述子セットR_1、R_2、R_3を含む。フォールバック構成Ｃ２は、アクティビティを実施するのに必要な対応するリソースを識別するための異なるリソース記述子セットR_1およびR_4を含む。新しいアクティビティＡ２は、１つの構成Ｃ１（番号１２０６として参照する）を有し、この構成Ｃ１は、対応するリソースを識別するための１つの記述子R_3を含む。

【０１１７】

この図示の例では、リソース記述子R_3は、番号１０４として参照するリソースプロバイダR_3を識別する。リソースプロバイダR_3は、チューナなど、一度しか割り振ることのできないリソースを制御するものとする。さらに、リソースプロバイダR_3によって維持されるカウンタ１２０８は、唯一のチューナが現在、優先度のより低いアクティビティＡ

１の既存の構成Ｃ１に割り振られていることを示すものとする。
【０１１８】
リソースマネージャ１０２が、新しいアクティビティＡ２を生み出したコンシューマから予約要求を受け取ったとき、リソースマネージャ１０２は、リソースプロバイダ１０４がもはや他方のアクティビティＡ１にリソースを割り振ることができないことを知る。したがってリソースマネージャは、優先度のより低いアクティビティＡ１をプリエンプトして、記述子R_3に関連するリソースを優先度のより高いアクティビティＡ２に移行すべきであると決定する。
【０１１９】
リソースマネージャは、優先度のより低いアクティビティＡ１が代替構成を有するかどうかを決定する（すなわち図１１のステップ１１００）。この場合、アクティビティＡ１は、プロバイダR_3によって制御されるプリエンプト対象のリソースを必要としないフォールバック構成Ｃ２を有する。したがってリソースマネージャ１０２は、フォールバック構成Ｃ２内のリソース（すなわちプロバイダR_1およびR_4から提供されるリソース）を予約し、アクティビティＡ１に関連するコンシューマに、プリエンプトされることおよびフォールバック構成Ｃ２に切り替える必要があることを通知する（すなわち図１１のステップ１１０８および１１１０）。
【０１２０】
アクティビティＡ１は、矢印１２１２で表すように、フォールバック構成Ｃ２の使用に移行する。リソースマネージャ１０２は、矢印１２１０で表すように、プロバイダR_3のリソースをアクティビティＡ１から新しいアクティビティＡ２に再割振りする。
【０１２１】
アクティビティＡ１に関連するコンシューマへの影響が小さく、かつフォールバック構成Ｃ２への変換から生じる分断なしにプロセスが連続できることが望ましい。例えば、ハードウェアビデオデコーダがなくなるがソフトウェアデコーダで置き換えることができる前述の状況を考えてみる。アプリケーションは、ソフトウェアビデオデコーダをロードおよび初期化してから、ビデオストリームが中断しないような形でこれをビデオストリーム中に統合する方式をとることができる。
【０１２２】
アクティビティ内の構成間の違いは、非常に様々である場合がある。元の構成と代替構成が、必要なリソースの点でかなり共通することもある。実際この違いは、競合するリソースが代替構成から除去されている点だけにあることがある。あるいは、構成は大きく異なることもある。
【０１２３】
（アップグレード通知）
前のセクションでは、システム中のコンシューマは、プリエンプトされた場合に、より望ましい構成からより望ましくない構成に移行することが必要な場合がある。アップグレード通知は、リソースが再び利用可能になったときにコンシューマがより望ましい構成にアップグレードできるようにすることによる、帰路に関係するものである。
【０１２４】
図１３に、動作条件の変化に伴ってより好ましくない構成からより好ましい構成に動的にアップグレードするプロセスを示す。このプロセスはリソースマネージャで実施されるものであり、このプロセスについて図２を参照しながら述べる。図示のプロセスでは、アプリケーションが、（１）リソースの予約を最初に要求したときに、その最高ランクの構成内の所望のリソースをすべて予約することができなかったため、または、（２）前にプリエンプトされてより低いランクの構成に移行したために、より望ましくない構成で稼動しているものとする。
【０１２５】
ステップ１３００および１３０２で、リソースマネージャ１０２は、既存のアクティビティを監視し、アクティビティが完了してそのリソースを解放したときを検出する。リソー

スが解放されたとき、リソースマネージャ１０２は、構成が現在予約されていないアクティビティおよび最良ではない構成が予約されているアクティビティを含め、すべてのアクティビティを調べて、より好ましいいずれかの構成にアップグレードすることができるかどうかを決定する（ステップ１３０４）。アップグレードできる場合は、リソースマネージャ１０２は、それらのアクティビティに関連するコンシューマにアップグレード通知を送る（ステップ１３０６）。アップグレード通知は、コンシューマのアクティビティ内のより望ましい（すなわちより高くランクされた）構成を現在利用可能なリソースで満たせることをコンシューマに知らせる。

【０１２６】

コンシューマは、現在使用している構成よりも望ましい構成が利用可能になったときにアップグレード通知を受け取るように、リソースマネージャに登録する。ただし、アップグレード通知はオプションであり、コンシューマはこれらを受け取るように登録しなくてもよい。

【０１２７】

コンシューマがアップグレード通知を受け取ると、コンシューマは、アップグレード要求をリソースマネージャに提出して、新しい構成のためのリソースを予約するように頼むことができる。リソースマネージャは、アップグレード要求を受け取ると、新しい構成の予約を試みる（ステップ１３０８）。一実装形態では、リソースマネージャは本質的に図３と同じプロセスを採用するが、例外として、アップグレードされるコンシューマは、２つの構成、すなわち現在所有している構成と新しい構成とを短時間だけ予約していることが許容される。２つの構成の予約を可能にすることにより、古い構成から新しい構成にスムーズに移行することができる。コンシューマは、新しい構成に移行した後で、古い構成を解放する。別の実装形態では、アップグレードされるコンシューマアプリケーションは、まず古い構成による処理をシャットダウンし、次いで処理を再開する前に新しい構成を予約することが必要な場合もある。

【０１２８】

ただし、コンシューマが新しい構成の予約に成功する保証はない。例えば、優先度のより高い他のアクティビティもまた同じリソースを請求している場合がある。リソースマネージャは、コンシューマがより望ましい構成にアップグレードしようとして失敗した場合に、既存の構成を維持することを保証する。

【０１２９】

より高ランクの構成とより低ランクの構成を含めて、構成は、アクティビティが予約済みのリソースを有するかどうかにかかわらずいつでもアクティビティに加えることができることに留意されたい。現在予約されているリソースを有するアクティビティに構成を追加する場合であって、この構成が現在の構成よりも高いランクにあり、このより高ランクの構成を満たすことができる場合は、リソースマネージャは、このより高ランクの構成が可能であることをコンシューマに知らせる即時のアップグレード通知を送ることになる。

【０１３０】

図１４に、より望ましい構成にアップグレードするプロセスにおけるシナリオの１つを示す。図１４には、リソースマネージャ１０２にある、既存のアクティビティＡ１（番号１４００（１）として参照する）、待機中のアクティビティＡ２（番号１４００（２）として参照する）、および終了するアクティビティＡ３（番号１４００（３）として参照する）を示す。終了するアクティビティＡ３の優先度が最も高く、その次に待機中のアクティビティＡ２、その次に既存のアクティビティＡ１の順である。終了するアクティビティＡ３は、そのタスクを完了し、記述子R_1およびR_2で示すそのリソースを解放しているところである。

【０１３１】

中間優先度のアクティビティＡ２は、記述子R_2で示すリソースを必要とする単一の構成Ｃ１を有する。中間優先度のアクティビティＡ２は、前に、優先度のより高いアクティビティＡ３に割り振られたこのリソースへのアクセスを得ることができなかったので、待機

中である。
【０１３２】
既存のアクティビティＡ１は、記述子R_1を含む好ましい構成Ｃ１と、記述子R_4を含むフォールバック構成Ｃ２とを有する。現在、記述子R_1で示すリソースが優先度のより高いアクティビティＡ３によって拘束されているので、既存のアクティビティは現在、フォールバック構成Ｃ２を使用している。
【０１３３】
高優先度のアクティビティＡ３が終了し、記述子R_1およびR_2に関連するリソースを解放したとき、リソースマネージャ１０２は、既存のアクティビティが新たに解放されたリソースの利益を得ることができるかどうかを決定する（すなわち図１３のステップ１３０４および１３０８）。この場合、アクティビティＡ１とＡ２の両方が利益を得ることができる。したがってリソースマネージャ１０２は、アクティビティＡ１およびＡ２にそれぞれ関連するアプリケーション３２（１）および３２（２）にアップグレード通知を送る（すなわちステップ１３０６）。アプリケーションは、アップグレードすることを選択した場合はアップグレード要求を返す。この例では、リソースマネージャ１０２はアクティビティＡ１の構成を、より好ましくない構成Ｃ２から、記述子R_1で示す解放されたリソースを利用する、より好ましい構成Ｃ１にアップグレードする。これは矢印１４０２で表される。リソースマネージャはまた、アクティビティＡ２のために記述子R_2に関連するリソースの予約も行う。
【０１３４】
（構成の構築）
前述のように、リソースまたはリソースプロバイダは、リソースを提供することに加えて、それ自体で、他のリソースプロバイダによって管理される他のリソースのコンシューマになることもある。例えば、ＵＳＢカメラリソースは、ＵＳＢドライバから提供される帯域幅のコンシューマである。したがって、ＵＳＢドライバは、ＵＳＢカメラリソースを親に持つ従属リソースプロバイダである。これにもかかわらず、アプリケーションは、ＵＳＢカメラを要求するとき、カメラがＵＳＢドライバのサービスを利用する必要があることを意識していないことがある。すなわち、アプリケーションなどのリソースコンシューマの中には、タスクを実施するのに必要な従属リソースのすべてを意識しているか知っているわけではないものもある。これらのタイプのリソースコンシューマは、そのタスクを実施するのに必要な「トップレベル」のリソースだけしか意識していないことがある。このタイプのリソースコンシューマの例として、後に開発される技術に先立って書かれるアプリケーションを考えてみる。具体的には、チューナリソースプロバイダからの「チューナリソース」だけを要求するようにテレビジョンアプリケーションを書く。チューナ技術における後の開発により、テレビジョンアプリケーションが元々書かれたときには利用可能でなかった追加のチューナが利用可能になる。これらの新しいチューナは、それ自体もアプリケーションが書かれた後に開発または改良されたものであるかもしれないリソースのコンシューマであることがある。それでもなお、テレビジョンアプリケーションが「チューナリソース」を要求するときは、より古いアプリケーションがより新しいチューナを使用することができるように、新しいチューナリソースおよびその従属リソースを１つ以上の構成内に組み込む方式が必要である。有利にも、述べた実施形態の態様では、各リソースプロバイダがそれ自体のリソースニーズを意識しているような形で、これを行うことができる。したがって、構成の構築段階の間にリソースプロバイダが呼び出されたとき、リソースプロバイダは、適当なすべてのステップを踏んで、構築された構成内でその従属リソースが適切に表されるようにすることができる。
【０１３５】
図１５に、リソースコンシューマの知らないリソースであるにもかかわらずリソースが構築中の構成内に表される方式の、ほんの１つを例示するアーキテクチャを示す。以下に提供する記述は、リソースコンシューマがタスクの実施に必要な従属リソースを意識していないという特定の場合に合わせたものであることを理解されたい。リソースコンシューマ

が、従属リソースプロバイダの１つまたはすべてを知っていることもあり得る。その場合、以下に述べる構成は、例えばリソースコンシューマからリソースプロバイダへのいくつかの直接呼出しを介して構築されることになる。しかしこの考察では、そのような場合ではないものとする。

【０１３６】

以下に述べる処理は、１つ以上の構成が生み出された後で実施され、ステップ３０６（図３）に対応する。ここでは、１つのアクティビティ１２２（１）内に１つの構成１２４（１）だけを示す。１つ以上のアクティビティ内に複数の構成を構築することもできることは明らかなはずである。この時点では、構成はポピュレートされていない。構成がポピュレートされるとき、アクティビティが関連している特定のタスクを実施するためのリソースに寄与する各リソースプロバイダにより、１つ以上のリソース記述子が構成に加えられる。各リソース記述子は、そのリソースプロバイダを識別し、特定の構成に必要な関連するリソースの量を示す。この具体例では、１つのリソースコンシューマだけが利用され、アプリケーション３２（１）を構成する。

【０１３７】

図示の例では、アプリケーション３２（１）は、そのタスクを実施するためにそれに必要なリソースのサブセット（したがってそれらに関連するリソースプロバイダ）だけしか意識していないかわかっていない。ここで、アプリケーション３２（１）は、リソースプロバイダ１０４（１）および１０４（２）を含む第１のリソースプロバイダのセットを意識している。アプリケーション３２（１）は、第２のリソースプロバイダのセット（すなわちリソースプロバイダ１０４（３）および１０４（４））は意識しておらず、わかっていない。これらのリソースプロバイダは、リソースプロバイダ１０４（２）によって使用され、リソースプロバイダ１０４（２）がタスクを実施するのを補助する。各リソースプロバイダ１０４（１）－１０４（４）は、構成１２４（１）がポピュレートされているときに他のリソースプロバイダを呼び出すように構成される。アプリケーション３２（１）は、いくつかのリソースプロバイダは意識しているので、それに応じてそれらのリソースプロバイダだけを呼び出し、したがってそれらのリソースプロバイダは、それらに関連するリソース記述子で構成１２４（１）をポピュレートするのを可能にする情報を提供することができる。アプリケーション３２（１）が意識していないリソースプロバイダは、それらのリソースを使用する他のリソースプロバイダによって呼び出される。

【０１３８】

例えば図１５では、アプリケーションは、リソースプロバイダ１０４（１）および１０４（２）を意識している。したがってアプリケーションは、構成をポピュレートまたは構築しようとするとき、これらのリソースプロバイダを直接呼び出す（矢印１５０１および１５０２で示す）。述べる実施形態では、リソースプロバイダに対して関数呼出しを行い、アクティビティを（例えばアクティビティＩＤまたはハンドルによって）識別し、ポピュレートされている構成を（例えば構成ハンドルまたはＩＤによって）識別する。これらの関数呼出しを受け取るのに応答して、各リソースプロバイダは、リソースマネージャ１０２に対して関数呼出しを行い、関連するリソースに関する情報を提供する。次いでリソースマネージャ１０２は、この情報を使用して構成を維持する。図示の例では、このような情報は、関連するリソース記述子の形で提供される。この例では、各リソースプロバイダ１０４（１）および１０４（２）は、それぞれリソース記述子R_2およびR_4を構築するのに必要な情報をリソースマネージャ１０２に提供する。これを、それぞれのリソースプロバイダから発してリソースマネージャ１０２で終わる矢印１５０３および１５０４で示す。しかし、アプリケーションはリソースプロバイダ１０４（３）および１０４（４）を意識していないので、これらは直接呼び出さない。そうではなくこの例では、これらのリソースを利用する親リソースプロバイダがこれらを呼び出す。したがって、リソースプロバイダ１０４（２）がリソースプロバイダ１０４（３）を呼び出し（矢印１５０６）、ポピュレートされている構成内のそのリソース記述子に関するアクティビティハンドルおよびリソース識別子をそれに渡す。この呼出しに応答して、リソースプロバイダ１０４（３

）は、リソースマネージャが構成を維持するのに必要な情報によって、リソースマネージャ１０２を呼び出す（矢印１５０８）（R_3で示す通信矢印で示すように）。この情報は、リソースプロバイダ１０４（２）に関するいわゆる関係情報を含む。この例では、関係情報は、リソースプロバイダ１０４（３）がリソースプロバイダ１０４（２）と有する子／親の関係を記述する。同様に、リソースプロバイダ１０４（３）はリソースプロバイダ１０４（４）に関連するリソースを使用するので、リソースプロバイダ１０４（３）は、ポピュレートされている構成内におけるそのリソース記述子に関するアクティビティハンドルおよびリソース識別子によって、リソースプロバイダ１０４（４）を呼び出す（矢印１５１０）。リソースプロバイダ１０４（４）は、リソースマネージャが構成を維持できるように、情報（その親を記述する関係情報を含む）によってリソースマネージャを呼び出す（矢印１５１２）。この通信を、R_1で示す通信矢印で示す。

【０１３９】

この例では、親／子の関係の性質が、いくつかのリソースの従属関係を記述する階層ツリーに編成されてこれを構成する記述子１２６で示されている。述べる構成は階層ツリーの形をとるが、必ずしも階層ツリーの形ではない構成を含めた他の構成を用いることもでき、例えば、平面構成や直線構成、またはリソース従属関係を記述しない構成を用いることができる。しかし、特定条件が発生する場合には、階層構造を利用するのが有利である。これらの特定条件の例には、リソース予約の失敗、またはリソースのプリエンプトの失敗が含まれる。これらのそれぞれについては、後でより詳細に論じる。

【０１４０】

図１６に、前述した実施形態による例示的なリソース管理方法におけるステップを示す。ステップ１６００で、リソースプロバイダを呼び出す。図示の例では、この呼出しは、リソースプロバイダによってサポートされるインタフェースに対して行い、１つ以上の構成を構築またはポピュレートすることを望むときに使用する。リソースプロバイダに対する呼出しは、リソースコンシューマ（アプリケーションやシステムコンポーネントなど）から来る可能性もあり、別のリソースプロバイダから来る可能性もある。ステップ１６０２で呼出しを受け取り、これに応答して、リソースプロバイダはそのリソースに関する情報を供給する。この情報は、ポピュレートされている構成を維持するのに使用される。ステップ１６０６で、タスクを実施するためにリソースプロバイダがいずれか他のリソースプロバイダに依存しているかどうかを決定する。リソースコンシューマが従属リソースプロバイダを意識していないシステムでは、この決定は個々のリソースプロバイダによって行われる。リソースコンシューマが従属リソースプロバイダを意識しているシステムでは、この決定は、リソースコンシューマと個々のリソースプロバイダのいずれかによって行うことができる。図１５の例では、リソースプロバイダ１０４（１）についての決定は否定となり、したがってステップ１６０８で、他のリソースプロバイダに対する追加の呼出しがあるかどうかを決定する。追加の呼出しがある場合は、ステップ１６０８はステップ１６０２に分岐し、前述のように継続する。特定の構成について追加の呼出しがこれ以上行われない場合は、ステップ１６１０で、次の構成またはアクティビティを得て前述のように処理を繰り返すか、または終了する。ステップ１６０６で１つ以上の従属リソースプロバイダがあると決定された場合は、ステップ１６１２で従属リソースプロバイダを呼び出す。したがって、リソースプロバイダが呼び出されたとき、リソースプロバイダは、他のどのリソースプロバイダを呼び出すかがわかるようにプログラムされる。図１５の例では、アプリケーションがリソースプロバイダ１０４（２）を呼び出したとき、このリソースプロバイダは、リソースプロバイダ１０４（３）を呼び出すことがわかる（ステップ１６０６）。したがって、リソースプロバイダ１０４（３）を呼び出し、ポピュレートされている構成内のそのリソース記述子に関するアクティビティハンドルおよびリソース識別子を渡す。この呼出しを受け取るのに応答して、リソースプロバイダ１０４（３）は、構成を構築またはポピュレートするのに使用されるそのリソースに関する追加の情報を供給する（ステップ１６１６）。次いでステップ１６１６はステップ１６０６に分岐し、追加の従属リソースがあるかどうか決定する。このステップは、リソースプロバイダ１０４（２

）によって実施される可能性もあり（１つ以上のリソースプロバイダに依存する場合）、またはリソースプロバイダ１０４（３）によって実施される可能性もある（１つのリソースプロバイダに依存する場合）。この特定の例では、リソースプロバイダ１０４（２）は１つのリソースプロバイダだけに依存するので、このステップはリソースプロバイダ１０４（３）によって行われる。この特定の場合、リソースプロバイダ１０４（３）はリソースプロバイダ１０４（４）に依存するので、処理はステップ１６１２－１６１６の中を進み、追加の呼出しがあるかどうかに応じてステップ１６０２または１６１０に進むことになる。

【０１４１】

図示の例では、異なる複数のリソースプロバイダから情報を受け取り、階層ツリーからなる構成の構築に利用する。ツリーはリソースマネージャ１０２によって維持され、予約およびエラー通知の発布に利用される。

【０１４２】

（エラー通知）

１つ以上のアクティビティに対して１つ以上の構成が構築された後、リソースマネージャ１０２は、これらのアクティビティのための構成を予約することを試みることができる。これを行うための例示的なプロセスについては、先に詳述してある。構成を予約しようとする途中で、１つ以上のリソースを予約することができないこともあり得る。この場合、予約は失敗し、リソースは利用不可能である。このことが生じたときは、予約失敗をリソースコンシューマに通知することが望ましい。この理由の１つは、リソースコンシューマが、通知を受けた後で代替構成の構築または予約を試みることができるからである。もう１つの理由は、そのタスクを実施することを可能にする様々なオプション、すなわち他のリソース設定をリソースコンシューマに提示することができるからである。

【０１４３】

しかし、失敗したまたはプリエンプトされた従属リソースをリソースコンシューマが意識しない実施形態では、通知はいくらかの困難を呈する。例えば、また図１５に関しては、リソースプロバイダ１０４（４）に関連するリソースを予約しようとする途中で予約が失敗した場合、リソースプロバイダ１０４（４）によって生成されるエラーメッセージは通常、リソースコンシューマには理解されない独自の用語を含むことになる。このため、理解可能な形でリソースコンシューマに明確に失敗を伝える方式が必要である。階層ツリー構造を利用する実施形態では、エラー報告は単に、リソースコンシューマが認識するリソースプロバイダが見つかるまでリソース従属関係の連鎖を辿ることによって達成される。次いで、認識されるリソースプロバイダを介してリソースコンシューマにエラーを報告する。このプロセスの間、最初に従属リソースプロバイダによって発布されたエラーメッセージは、その独自の形式でその親リソースプロバイダによって受け取られ、次の順番のリソースプロバイダに理解される形式に変換され、しかるべく転送される。最終的に、リソースコンシューマには、リソースコンシューマに理解される用語で表されたエラー通知が届く。

【０１４４】

例として、次の場合を考えてみる。アプリケーションが、３０フレーム／秒で再生できるようにＵＳＢカメラリソースを予約しようとする。カメラリソースは利用可能かもしれないが、３０フレーム／秒に対応するのに十分なＵＳＢ帯域幅がないと仮定する。生成される失敗は、ＵＳＢ帯域幅リソースプロバイダによって生成され、アプリケーションに理解されない独自の用語で定義される。このため、ＵＳＢ帯域幅プロバイダからの失敗は、その親すなわちＵＳＢカメラリソースプロバイダに報告される。ＵＳＢカメラリソースプロバイダは、失敗を、アプリケーションに報告される形に変換する。

【０１４５】

リソースコンシューマへのエラー報告は、様々な形をとることができる。例えば、エラー報告は単に、知られているリソース、例えばカメラリソースが現在利用可能でないことを報告するものであることがある。あるいは、エラー報告は、失敗の場合にリソースコンシ

ューマに１つ以上のオプションを提示することもできる。例えば、ＵＳＢ帯域幅の不足のせいでＵＳＢカメラリソースが失敗した場合、ＵＳＢ帯域幅リソースプロバイダは、それが有する利用可能な帯域幅をＵＳＢカメラプロバイダに提供することができる。次いでＵＳＢカメラリソースプロバイダは、この情報をアプリケーションに理解される形式に変換し、フレームレート低下（例えば５フレーム／秒）またはウィンドウサイズ縮小のオプションをアプリケーションに提供することができる。この例では、リソースの割当てが異なっていても、タスクはまだ実行できるであろう。以上の例は予約失敗に関して提供したものだが、リソースが別のアクティビティによる使用のためにプリエンプトされるときにも、同じエラー報告の例があてはまる。

【０１４６】

ただし、エラー報告は別の影響を有することもある。具体的には、リソースプロバイダの中には、発生して失敗を引き起こす恐れのある特定の問題に対して修復またはトラブルシューティングを試みるようにプログラムすることができるものもある。この場合、エラーをリソースコンシューマに報告する必要はない。トラブルシューティングは、従属関係の連鎖の中のどの地点でも行うことができる。例えば、ハードウェアデバイスを管理するリソースプロバイダを考えてみる。このハードウェアデバイスは、プリエンプトされたせいで利用可能でないと仮定する。リソースプロバイダは、エラー報告をリソースコンシューマに発布するのではなく、リソースコンシューマには実際のハードウェアデバイスであるかのように見える「仮想デバイス」を提供することができる。リソースコンシューマはこの場合、実際のハードウェアデバイスが利用可能になるまで、そうとは知らずに仮想デバイスを使用することになる。このようにすれば、リソースコンシューマにエラーは報告されず、タスクは依然として実施することができる。

【０１４７】

階層ツリー構成は暗黙的に順序付けを含むので、階層ツリー構成を用いることにより、エラー報告は、平面構成や線形構成よりもずっと効率的になる。平面構成や線形構成を用いることも可能だが、望ましさは劣る。

【０１４８】

図１７に、述べた実施形態によるエラー報告方法におけるステップを記述する。ステップ１７００で、リソースを階層ツリー構造として表す。これをどのように行うことができるかについての具体的な例および例示的な構造は、先に「構成の構築」のセクションで挙げたものである。ステップ１７０２で、特定のリソースに関連する特定条件を検出する。この例では、特定条件はエラー条件であり、リソースマネージャ１０２は、予約失敗またはプリエンプトがあるときがわかるようにプログラムされているので、ステップ１７０２はリソースマネージャ１０２（図１５）によって行うことができる。これは、リソースマネージャがリソース計算を実行し、リソース競合を決定するからである。次いでステップ１７０４で、リソースに関連するリソースプロバイダに通知する。これを行うために、リソースマネージャは各プロバイダにコールバックする（すなわちプロバイダは登録時にコールバックを設定する）。リソースマネージャは現エラーパケット（最初は空）をプロバイダに与え、次いで、変換されたパケットをプロバイダから反対に受け取る。次いで、この変換されたパケットを連鎖の中の次のプロバイダに渡し、以下同様にする。プロバイダのうちの１つがそれ自体でエラーを修復する場合もあり、この場合は、それ以上連鎖を上ってパケットを転送する必要はない。

【０１４９】

（ポリシーマネージャ）

ポリシーマネージャは、リソースマネージャと共に、複数のアプリケーションが同じリソースを求めて争うときにどのアプリケーションが限られたリソースにアクセスして使用することができるかを決定する。アプリケーション自体が自発的にリソースを開始および利用することはなく、アプリケーションがリソースによって管理されるアクティビティの優先度を制御することもない。そうではなく、リソースは、ポリシーマネージャにおいて確立されたポリシーに基づいてリソースマネージャによって割り振られる。

【０１５０】
ポリシーには様々なタイプが考えられる。例えばあるポリシーセットは、どのアプリケーションおよび／またはユーザが他を制してリソース使用の優先度を有するかに基づいてリソース割振りを決定する優先度ベースの競合解決と共に使用することができる。用語「優先度」は、従来のスレッド優先度やプロセス優先度の意味では用いず、同じリソースを求めて争うコンシューマ間の調停ポリシーのコンテキストで用いる。優先度は、個々のリソースに割り当てられるのではなく、アプリケーションがリソースマネージャに確立したアクティビティに割り当てられる。
【０１５１】
ポリシーは、どのアクティビティが他のアクティビティと比較して何らかの形で「より重要」かまたは「より重要でない」かを（例えばユーザにとって「より重要」）決定する。これによってリソース管理アーキテクチャは、所望の、ただし限られたリソースを、より重要でないアクティビティからより重要なアクティビティに移行することができる。
【０１５２】
別の実行可能なポリシーは、「先着予約順」である。このポリシーに従うことにより、リソースマネージャは、アクティビティのためのリソースを先着順で予約することになる。
【０１５３】
別の可能なポリシーは、「一番最近の予約順」である。このポリシーを使用すると、リソースマネージャは、一番最近に予約を求めたアクティビティのためのリソースを予約することを試みる。
【０１５４】
別の可能なポリシーには、何らかのタイプのバランスガイドラインまたは提携ガイドラインを達成するためのリソース共用や、ユーザがより重要なアクティビティを選ぶユーザ指定勝利者などが含まれる。
【０１５５】
図２に示すリソース管理アーキテクチャ１００は、別個かつ独立のポリシーマネージャ１０８によって確立されたポリシーに基づいてリソース割振り決定を行うリソースマネージャ１０２を実装する。ポリシーマネージャ１０８は、すべてのアクティビティにリソースを割り振ることができないような競合があるときにどのアクティビティがリソースへのアクセスを得るべきかを、アプリケーションから独立して決定する。システムまたはユーザがポリシー１１０を設定し、ポリシーマネージャ１０８がこれらを絶対的な優先度に変換する。
【０１５６】
図１８に、ポリシーマネージャ１０８をリソース管理アーキテクチャ１００と共に利用する場合を示す例示的なポリシー管理アーキテクチャ１８００を示す。リソースマネージャ１０２は、定義済みのＡＰＩ（アプリケーションプログラムインタフェース）１２０を公開して、リソースを求める要求をアプリケーション３２（１）－３２（Ａ）から受け入れる。後の「リソースマネージャＡＰＩ」の項目で、ＡＰＩの１つについて詳細に述べる。アプリケーション３２は、タスクを実施したいとき、ＡＰＩ１２０を使用してリソースマネージャ１０２にアクティビティ１２２を生み出す。アクティビティは、システム中で実施されるタスクに関連するデータ構造である。実施されるタスク１つにつき１つのアクティビティが存在する。図では、リソースマネージャ１０２は、アプリケーション３２（１）－３２（Ａ）によって生み出されたアクティビティ１２２（１）－１２２（Ｎ）を含む。
【０１５７】
ポリシー管理アーキテクチャ１８００はソフトウェアによって実現され、この例では、ユーザレベルとカーネルレベルの両方のコンポーネントを有するポリシーマネージャ１０８を含む。ポリシーマネージャ１０８は、ユーザコンポーネント１８０２、カーネルコンポーネント１８０４、および対話バッファコンポーネント１８０６を有する。ポリシーマネージャ対話バッファ１８０６は、ポリシーマネージャカーネルコンポーネント１８０４と

ポリシーマネージャユーザコンポーネント１８０２との間の通知を保持する。
【０１５８】
ポリシーマネージャユーザインタフェース１８０８がユーザレベルにあるが、これはポリシーマネージャ１０８の外部にある。ポリシーは、最初にシステム内で生み出されるが、フレキシブルに設計され、したがってユーザは、ポリシーをカスタマイズすることにより、かつポリシーがポリシーマネージャ１０８によってどのように解釈されて使用されるかを調整することにより、システムを最適化することができる。ユーザはユーザインタフェース１８０８を介して、ポリシーを規定することができ、また、同じリソースを求めて争うアプリケーション間のリソース予約競合を解決するためにポリシーを適用する際の順番を確立することができる。
【０１５９】
ポリシーマネージャカーネルコンポーネント１８０４は、アクティビティ優先度および優先度修正を制御するための、ポリシーマネージャ１０８とリソースマネージャ１０２との間のインタフェースである。ポリシーマネージャカーネルコンポーネント１８０４は、リソース管理によって定義されたコールバックオブジェクトを開き、コールバックルーチンを登録する。リソースマネージャ１０２は、コールバックオブジェクトを使用して、リソースマネージャ１０２におけるアクティビティイベントをポリシーマネージャ１０８に通知する。
【０１６０】
ポリシーマネージャユーザコンポーネント１８０２は、３つのコンポーネントによって実現される。すなわち、（１）前述のポリシーコンポーネント１１０、（２）ポリシーマネージャディスパッチエンジン１８１０、および（３）アクティビティリスト１８１２である。ポリシーコンポーネント１１０は、リソース割振り決定を行うのに使用されるポリシーと、リソース割振り競合を解決するのに使用されるポリシーとを維持する。ポリシーコンポーネント１１０中で維持されるポリシーには、固定優先度ポリシー１８１４、フォーカスベースのポリシー１８１６、およびユーザ解決ポリシー１８１８が含まれる。ポリシーは、リソース競合を解決するために、ポリシー列挙数字（１）－（３）で示す順で適用する。すなわち、（１）として列挙した固定優先度ポリシー１８１４を使用してリソース競合を解決することができない場合は、次いでポリシー（２）のフォーカスベースのポリシー１８１６を適用し、以下同様にする。
【０１６１】
ポリシーは、アプリケーション３２が提供するアクティビティ特有の情報を利用して、システムアクティビティの優先度を決定する。アプリケーション３２は、アクティビティに関するポリシー属性を設定することにより、この情報を提供する。この情報は、ポリシー属性を生成、削除、修正、検索するＡＰＩセットを介して提供する。このＡＰＩセットは、後で「リソースマネージャＡＰＩ」の項目で詳細に述べるＲＭＵ ＡＰＩを拡張したものである。具体的なＡＰＩセットについては、後で「ＲＭＵ ＡＰＩの拡張」の項目で述べる。
【０１６２】
リソースマネージャ１０２は、アクティビティ１２２が生成または破壊されたとき、およびリソースがアクティビティ構成のために予約または予約解除されたとき、アクティビティイベントをポリシーマネージャ１０８に通知する。ポリシーマネージャ１０８はまた、アクティビティ間にリソース予約競合があるとき、およびユーザ対話式アプリケーションのプロセスがシステムフォーカスを得たときにも通知を受ける。
【０１６３】
ポリシーマネージャディスパッチエンジン１８１０は、リソースマネージャ１０２からポリシーマネージャカーネルコンポーネント１８０４および対話バッファコンポーネント１８０６を介してアクティビティイベント通知を受け取り、次いでこれらの通知を、次の操作のためにポリシー１１０にディスパッチする。ポリシー１１０によってアクティビティが「格付け」された後で、ポリシーマネージャディスパッチエンジン１８１０は、絶対的

なアクティビティ優先度を決定する。ポリシーマネージャディスパッチエンジン１８１０はまた、すべてのポリシーのリスト、ならびに、各アクティビティの優先度を関連させたアクティビティリスト１８１２も維持する。

【０１６４】

より具体的には、ポリシーマネージャディスパッチエンジン１８１０は、アプリケーションがメソッドＲＭＣｒｅａｔｅＡｃｔｉｖｉｔｙ、ＲＭＤｅｓｔｒｏｙＡｃｔｉｖｉｔｙ、ＲＭＲｅｓｅｒｖｅＲｅｓｏｕｒｃｅｓ、ＲＭＵｎｒｅｓｅｒｖｅＲｅｓｏｕｒｃｅｓを呼び出したとき、リソースマネージャ１０２からアクティビティイベント通知を受け取る。イベント通知を受け取ると、ポリシーマネージャディスパッチエンジン１８１０は、アクティビティリスト１８１２を更新し、ポリシー１１０に通知を送る。ポリシー１１０は、システム中の現アクティビティの相対的な重要度順序付けに従ってアクティビティリストの優先度が再決定されるように、アクティビティデータ構造の優先度の再決定をトリガすることができる。

【０１６５】

アクティビティリスト１８１２は、現アクティビティのリストをポリシー１１０に渡してアクティビティの優先度が順序付けされるようにするために、ポリシーマネージャディスパッチエンジン１８１０によって使用されるオブジェクトである。アクティビティリスト１８１２は、すべてのアクティビティ１２２を含むアクティビティ情報オブジェクトの集合として渡される。ポリシー１１０は、アクティビティ情報オブジェクトの集合を、優先度順のアクティビティサブセットに修正する。ポリシー１１０がアクティビティ１２２の格付けを完了した後でポリシーマネージャディスパッチエンジン１８１０が絶対的なアクティビティ優先度を決定するとき、同じサブセット中のアクティビティは、同じ絶対的な優先度を受け取る。

【０１６６】

固定優先度ポリシー１８１４は、アクティビティ範疇に関するユーザ定義の重要度順序付けに基づいてアクティビティ優先度を決定する。例えば、ユーザは、特定の重要度順序を有するものとして以下のアクティビティ範疇を定義することができる。

【０１６７】

既存のアクティビティ	範疇	範疇の重要度
Ａ１　ＤＶＤ$_1$の視聴	Ｃ１	1
Ａ２　ＴＶの視聴	Ｃ２	2
Ａ３　ＤＶＤ$_2$の視聴	Ｃ３	3
新しいアクティビティ	範疇	範疇の重要度
Ａ４　録画	Ｃ２	2

【０１６８】

アクティビティＡ１－Ａ３は、リソースマネージャ１０２に存在している。新しいアクティビティＡ４が生み出されたとき、リソースマネージャ１０２は、アクティビティイベントをポリシーマネージャ１０８に通知する。ポリシーマネージャディスパッチエンジン１８１０は、アクティビティリスト１８１２を更新し、アクティビティ１２２の優先度が順序付けされるように現アクティビティ１２２のリストをポリシー１１０に渡す。固定優先度ポリシー１８１４は、アクティビティリスト１８１２を、優先度順にアクティビティのサブセットに修正する。

【０１６９】

優先度セット	アクティビティ
1	Ａ１
2	Ａ２、Ａ４
3	Ａ３

【０１７０】

フォーカスペースのポリシー１８１６は、リソース管理アクティビティ１２２を生み出したプロセスのフォーカス履歴に基づいてアクティビティ優先度を決定する。ユーザ対話式

アプリケーションだけが、フォーカス履歴を維持するためにフォーカスベースを必要とする。したがって、フォーカスベースのポリシー１８１６は、ユーザ対話式アプリケーションによって生み出されたアクティビティ１２２の優先度を決定する。ユーザ対話式アプリケーションのプロセスがフォーカスを得たとき、このプロセスのアクティビティは最初にフォーカス履歴中で識別される。

【０１７１】

以上の例から、ユーザは、アクティビティＡ１－Ａ４を特定の重要度順序を有するものとして定義し、その結果、固定優先度ポリシー１８１４は、各アクティビティがどの優先度セットに属するかを決定した。ユーザがＴＶの視聴（アクティビティＡ２、優先度２）を開始し、その後、録画（アクティビティＡ４、優先度２）を行うことが予定された場合、リソースマネージャ１０２は、システムの唯一のチューナリソースをどちらのアクティビティに割り振るべきかを決定しなければならない。フォーカスベースのポリシー１８１６は、ユーザがＴＶの視聴を開始したとき、ＴＶからの画像を表示するプロセスを、フォーカス履歴中の最初のものとして識別することになる。したがって以下のように、アクティビティＡ２が、優先度セット２でフォーカス優先度を有することになる。

【０１７２】

【表１】

優先度セット	アクティビティ
1	A1
2	A2, A4
3	A3

【０１７３】

ユーザ解決ポリシー１８１８は、リソースマネージャ１０２が現在のアクティビティ優先度に基づいて競合を解決することができないときに、リソース予約競合を解決する。例えば、ユーザは以下のように、同時に開始する異なる２つのチャネルを録画するために、予定された２つの録画アクティビティを有する場合がある。

【０１７４】

既存のアクティビティ	範疇	範疇の重要度
Ａ１　録画Ａ	Ｃ１	1
Ａ２　録画Ｂ	Ｃ１	1

【０１７５】

最初に、固定優先度ポリシー１８１４がアクティビティ優先度を決定し、ポリシー１１０が、アクティビティリスト１８１２をアクティビティの優先度サブセットに修正することになる。

【０１７６】

優先度セット	アクティビティ
1	Ａ１、Ａ２

【０１７７】

アクティビティＡ１とＡ２が両方ともユーザにとって同じ優先度を有すると仮定すると、リソースマネージャ１０２がシステムの唯一のチューナリソースをどちらのアクティビティに割り振るべきか決定しなければならないときに、リソース競合が発生する。録画Ａと録画Ｂはいずれもシステムフォーカスを得たプロセスではないので、フォーカスベースのポリシー１８１６はこの競合を解決することはできない。

【０１７８】

ユーザ解決ポリシー１８１８は、アクティビティＡ１とＡ２の間で競合が存在するという情報をポリシーマネージャディスパッチエンジン１８１０から受け取ると、競合を解決するためにポリシーマネージャユーザインタフェース１８０８を介してユーザと通信する。

ユーザがアクティビティの優先度を再決定すると、ポリシーマネージャのポリシー１１０は、ユーザの解決選択を反映するようにアクティビティリスト１８１２を修正する。ユーザ解決ポリシー１８１８はまた、リソース競合を解決するためのユーザ対話の必要を減らすために、すべてのアクティビティについてそれらの存続期間にわたるユーザ解決履歴も維持する。

【０１７９】

図１９に、リソースマネージャ１０２において生み出されたアクティビティのポリシー管理の方法におけるステップを示す。ステップ１９００で、ポリシーマネージャディスパッチエンジン１８１０は、リソースマネージャ１０２からアクティビティ通知を受け取る。ポリシーマネージャ１０８は、アクティビティ生成、アクティビティ破壊、アクティビティ予約、アクティビティ予約解除、およびリソース予約競合に関して通知を受ける。

【０１８０】

アクティビティ通知を受け取ると、ポリシーマネージャディスパッチエンジン１８１０は、アクティビティリスト１８１２を更新する（ステップ１９０２）。次いでポリシーマネージャディスパッチエンジン１８１０は、アクティビティ通知をポリシー１１０に転送する（ステップ１９０４）。

【０１８１】

ステップ１９０６で、ポリシー１１０は、アクティビティ通知（ステップ１９００）が２つ以上のアクティビティ間のリソース予約競合に関するものであったかどうかを決定する。ポリシーが競合を解決する必要がある場合は（すなわちステップ１９０６からの「ｙｅｓ」ブランチ）、ポリシーは、競合するアクティビティの相対的な重要度を決定する（ステップ１９０８）。これは、前述のように、ユーザ解決ポリシー１８１８を介してユーザと相談することを含む場合がある。

【０１８２】

競合するアクティビティの相対的な重要度を決定した後は（ステップ１９０８）、またはポリシーが競合を解決する必要がない場合は（すなわちステップ１９０６からの「ｎｏ」ブランチ）、ポリシーは、アクティビティの優先度の再決定が必要かどうかを決定する（ステップ１９１０）。通常、アクティビティ生成およびアクティビティ予約の場合は、アクティビティの優先度の再決定が必要である。アクティビティ破壊またはアクティビティ予約解除の場合は、通常はアクティビティの優先度の再決定は必要ない。

【０１８３】

アクティビティの優先度の再決定が必要ない場合は（すなわちステップ１９１０からの「ｎｏ」ブランチ）、ポリシーマネージャ１０８は、リソースマネージャ１０２のアクティビティ優先度を更新する必要はなく、方法は終了する（ステップ１９１２）。アクティビティの優先度の再決定が必要な場合は（すなわちステップ１９１０からの「ｙｅｓ」ブランチ）、ポリシー１１０はアクティビティの優先度を再決定する（ステップ１９１４）。アクティビティの優先度が再決定された後、ポリシーマネージャディスパッチエンジン１８１０は、アクティビティの絶対的な優先度を決定する（ステップ１９１６）。次いでポリシーマネージャディスパッチエンジン１８１０は、リソースマネージャ１０２のアクティビティ優先度を更新する（ステップ１９１８）。

【０１８４】

以上の例では、ポリシーが優先度ベースの競合解決と協働すると仮定している。しかし他の状況では、リソースコンシューマ間に優先度がないこともある。すなわち、アプリケーションに関連する各アクティビティが同一の優先度を有するか、優先度を全く有さないことがある。

【０１８５】

以下のシナリオを考えてみる。アクティビティＡ１は、２つの構成Ｃ１およびＣ２を有する。最も好ましい構成Ｃ１は、２つのリソースR_1およびR_2を必要とする。より好ましくない構成Ｃ２は、１つのリソースR_1だけを利用する。アクティビティＡ２は、２つの構成、すなわち（１）２つのリソースR_1およびR_2を利用する最も好ましい構成Ｃ１、

および（２）１つのリソースR_2だけを利用するより好ましくない構成Ｃ２を有する。

【０１８６】

アクティビティＡ１とＡ２の好ましい構成を同時に実行しようとすると、競合が生じる。この状況では、リソースマネージャは、両方のアクティビティについて第２の構成を実行することを選択することができ、それにより２つのアクティビティを同時に継続することができる。アクティビティＡ１とＡ２の間には優先度の差がないので、一方を犠牲にして他方を実行する必要はない。

【０１８７】

別のポリシーセットは、より少ないアクティビティをそれらの最も好ましい構成で実行するのではなく、できるだけ多くのアクティビティを、各アクティビティがより好ましくない構成を使用する形で実行することに関するものとすることができる。例えば、前述のシナリオで、アクティビティＡ１がアクティビティＡ２よりも優先度が高いとしても、ポリシーマネージャは、アクティビティＡ１だけをその最良の構成で実施するのではなく、両方のアクティビティをそれらの第２の構成で実行する方を好む。

【０１８８】

以下は、ポリシー管理アーキテクチャと共に利用する例示的なメソッドである。

【０１８９】

（ＣＰＯＬＩＣＹＭＡＮＡＧＥＲクラス中のメソッド）

１．ｖｏｉｄ　Ｒｅｐｒｉｏｒｉｔｉｚｅ（）

ａ）説明

このメソッドは、ポリシーがアクティビティの優先度の再決定をトリガすることができるように、主ポリシーマネージャディスパッチエンジン（ＣＰｏｌｉｃｙＭａｎａｇｅｒ）によって公開される。例えば、リソースマネージャ中にアクティビティを生み出したアプリケーションにフォーカスが変化したときは、フォーカスポリシーがこのメソッドを使用して、アクティビティの優先度の再決定をトリガする。このメソッドは、ポリシーマネージャディスパッチエンジン中に通知をポストしてから戻る。この通知の結果、ポリシーマネージャディスパッチエンジンはＣＰｏｌｉｃｙＭａｎａｇｅｒ：：ＣａｌｃｕｌａｔｅＰｒｉｏｒｉｔｉｅｓ（）を呼び出し、これは、優先度の再決定のためにアクティビティリストをポリシーに渡す。

ｂ）戻り値

なし。

２．ＨＲＥＳＵＬＴ　ＣａｌｃｕｌａｔｅＰｒｉｏｒｉｔｉｅｓ（）

ａ）説明

このメソッドは、システム中のすべてのアクティビティについて優先度を計算するのに使用する。このメソッドはＣＰｏｌｉｃｙ：：ＣａｌｃｕｌａｔｅＰｒｉｏｒｉｔｉｅｓ（）を呼び出す。得られるコンパートメント化されたアクティビティリストは、次いでアクティビティの絶対的な優先度を計算するのに使用され、次に、この絶対的な優先度がリソースマネージャ中に設定される。

ｂ）戻り値

Ｓ＿ＯＫ　成功の場合。

Ｅ＿ＦＡＩＬ　そうでない場合。

３．ＣＡｃｔｉｖｉｔｙＬｉｓｔ＊ＧｅｔＡｃｔｉｖｉｔｙＬｉｓｔ（）

ａ）説明

このメソッドは、ポリシーがシステム中の現アクティビティのリストを得るのに使用する。例えば、フォーカスベースのポリシーが、関連するリソースマネージャアクティビティを有するプロセスについてのフォーカス履歴を維持するのを開始することができるように、初期化してシステム中の現在のアクティビティリストを得るときに使用する。

ｂ）戻り値

システム中のアクティビティを列挙できるようなメソッドを公開するＣＡｃｔｉｖｉｔｙＬｉｓｔオブジェクトへのポインタ。

4．ｖｏｉｄ ＯｎＮｅｗＡｃｔｉｖｉｔｙ（ＰＲＭ＿ＡＣＴＩＶＩＴＹ Ａｃｔｉｖｉｔｙ）
ａ）説明
このメソッドは、ポリシーマネージャディスパッチエンジンが、新しいアクティビティが生み出されたことの通知をリソースマネージャから受け取ったときに、ポリシーマネージャディスパッチエンジンによって呼び出される。このメソッドは、リソースマネージャ中に生み出されたアクティビティを反映する新しいＣＡｃｔｉｖｉｔｙオブジェクトを生み出し、次いでこの通知を、ＣＰｏｌｉｃｙ：：ＯｎＮｅｗＡｃｔｉｖｉｔｙ（）を呼び出すことによってすべてのポリシーに通知する。次いでポリシーは、この通知を処理し、必要なら優先度の再決定をトリガすることができる。
ｂ）パラメータ
Ａｃｔｉｖｉｔｙ一付録に定義するＰＲＭ＿ＡＣＴＩＶＩＴＹタイプの構造体。この構造体は、アクティビティハンドルおよび所有プロセスＩＤを含む。
ｃ）戻り値
なし。
5．ｖｏｉｄ ＯｎＤｅｓｔｒｏｙＡｃｔｉｖｉｔｙ（ＰＲＭ＿ＡＣＴＩＶＩＴＹ Ａｃｔｉｖｉｔｙ）
ａ）説明
このメソッドは、ポリシーマネージャディスパッチエンジンが、アクティビティが破壊されたことの通知をリソースマネージャから受け取ったときに、ポリシーマネージャディスパッチエンジンによって呼び出される。この通知は、ＣＰｏｌｉｃｙ：：ＯｎＤｅｓｔｒｏｙＡｃｔｉｖｉｔｙ（）を使用してすべてのポリシーに渡される。アクティビティはこれを使用して、それらの非公開情報をクリーンアップするか、必要なら優先度の再決定をトリガすることができる。
ｂ）パラメータ
Ａｃｔｉｖｉｔｙ一付録に定義するＰＲＭ＿ＡＣＴＩＶＩＴＹタイプの構造体。この構造体は、アクティビティハンドルおよび所有プロセスＩＤを含む。
ｃ）戻り値
なし。
6．ｖｏｉｄ ＯｎＲｅｓｅｒｖｅ（ＰＲＭ＿ＡＣＴＩＶＩＴＹ Ａｃｔｉｖｉｔｙ）
ａ）説明
このメソッドは、ポリシーマネージャディスパッチエンジンが、アクティビティ構成についての予約が完了したことの通知をリソースマネージャから受け取ったときに、ポリシーマネージャディスパッチエンジンによって呼び出される。この通知は、ＣＰｏｌｉｃｙ：：ＯｎＲｅｓｅｒｖｅ（）を使用してすべてのポリシーに渡される。
ｂ）パラメータ
Ａｃｔｉｖｉｔｙ一付録に定義するＰＲＭ＿ＡＣＴＩＶＩＴＹタイプの構造体。この構造体は、アクティビティハンドルおよび所有プロセスＩＤを含む。
ｃ）戻り値
なし。
7．ｖｏｉｄ ＯｎＵｎｒｅｓｅｒｖｅ（ＰＲＭ＿ＡＣＴＩＶＩＴＹ Ａｃｔｉｖｉｔｙ）
ａ）説明
このメソッドは、ポリシーマネージャディスパッチエンジンが、アクティビティ構成についての予約解除が完了したことの通知をリソースマネージャから受け取ったときに、ポリシーマネージャディスパッチエンジンによって呼び出される。この通知は、ＣＰｏｌｉｃｙ：：ＯｎＵｎｒｅｓｅｒｖｅ（）を使用してすべてのポリシーに渡される。
ｂ）パラメータ
Ａｃｔｉｖｉｔｙ一付録に定義するＰＲＭ＿ＡＣＴＩＶＩＴＹタイプの構造体。この構造体は、アクティビティハンドルおよび所有プロセスＩＤを含む。
ｃ）戻り値

なし。
8．ＨＲＥＳＵＬＴ ＯｎＣｏｎｆｌｉｃｔ（ＰＲＭ＿ＡＣＴＩＶＩＴＹ ｃｏｎｆｌｉｃｔｉｎｇＡｃｔｉｖｉｔｙ，ＰＲＭ＿ＡＣＴＩＶＩＴＹ＊ｖｉｃｔｉｍＡｒｒａｙ，ＵＬＯＮＧ ｕｌＮｕｍＶｉｃｔｉｍｓ）
ａ）説明
これは、リソースマネージャが現在のアクティビティ優先度を用いてリソース競合を解決することができないときに呼び出される。次いでこのメソッドは、ＣｃｏｎｆｌｉｃｔＰｏｌｉｃｙ：：ＯｎＣｏｎｆｌｉｃｔ（）を呼び出して、競合に対する解決を得る。競合が解決されると、ポリシーマネージャディスパッチエンジンによって新しいアクティビティ優先度がリソースマネージャ中に設定される。
ｂ）パラメータ
ｃｏｎｆｌｉｃｔｉｎｇＡｃｔｉｖｉｔｙ－これは、予約時にリソース競合を引き起こしたアクティビティである。これは、付録に定義するＰＲＭ＿ＡＣＴＩＶＩＴＹタイプの構造体である。この構造体は、アクティビティハンドルおよび所有プロセスＩＤを含む。
ＶｉｃｔｉｍＡｒｒａｙ－犠牲者アクティビティであるＰＲＭ＿ＡＣＴＩＶＩＴＹ構造体の配列。
ＵＩＮｕｍＶｉｃｔｉｍｓ－犠牲者アクティビティの数。
ｃ）戻り値
Ｓ＿ＯＫ　ポリシーによって競合が解決される場合。
Ｅ＿ＦＡＩＬ　そうでない場合。
【０１９０】
（ＣＢＡＳＥＰＯＬＩＣＹクラス中のメソッド）
１．ｖｉｒｔｕａｌ ＨＲＥＳＵＬＴ ＧｅｔＰｏｌｉｃｙＮａｍｅ（ＬＰＴＳＴＲ＊ｐＢｕｆｆｅｒ，ＵＬＯＮＧ＊ｐＢｕｆＳｉｚｅ）＝０
ａ）説明
ポリシーについての名前を得る。
ｂ）パラメータ
ＰＢｕｆｆｅｒ－ｕｌＢｕｆＳｉｚｅで指定されたサイズの、名前のコピー先であるバッファへのポインタ。
ｐＢｕｆＳｉｚｅ－バッファのサイズ。ポリシー名のサイズがここにコピーされる。
ｃ）戻り値
Ｓ＿ＯＫ　成功の場合。
Ｅ＿ＯＵＴＯＦＭＥＭＯＲＹ　十分なメモリが利用可能でない場合。この場合、必要とされるサイズがｐＢｕｆＳｉｚｅにコピーされる。
Ｅ　ＦＡＩＬ　その他の場合。

```
2.virtual void OnNewActivity(CActivity*pActivity)
  virtual void OnDestroyActivity(CActivity*pActivity)
  virtual void OnRserveActivity(CActivity*pActivity)
  virtual void OnUnreserveActivity(CActivity*pActivity)
```

ａ）説明
これらは、ポリシーマネージャディスパッチエンジン（ＣＰｏｌｉｃｙＭａｎａｇｅｒ）がリソースマネージャから通知を受け取ったときに、ポリシーマネージャディスパッチエンジンによって呼び出される通知メソッドである。ポリシーは、これらの通知を受け取ると、ＣＰｏｌｉｃｙＭａｎａｇｅｒ：：Ｒｅｐｒｉｏｒｉｔｉｚｅを呼び出すことによって優先度の再決定をトリガすることができる。ポリシーはまた、これらの通知を使用して非公開情報を更新することもできる。例えば、フォーカスポリシーは、そのＯｎＮｅｗＡｃｔｉｖｉｔｙ（）が呼び出されたときに、そのフォーカス履歴追跡に新しいアクティビティを追加する。
ｂ）パラメータ
ＰＡｃｔｉｖｉｔｙ：アクティビティハンドルと、所有プロセスＩＤと、その他のアクテ

ィビティ特有情報とを含むＣＡｃｔｉｖｉｔｙオブジェクトへのポインタ。
ｃ）戻り値
なし。
【０１９１】
（ＣＰＯＬＩＣＹクラス中のメソッド（ＣＢａｓｅＰｏｌｉｃｙ中のメソッドに加えて））
１．ｖｉｒｔｕａｌ ＨＲＥＳＵＬＴ ＣａｌｃｕｌａｔｅＰｒｉｏｒｉｔｉｅｓ（ＣＢｕｃｋｅｔＬｉｓｔ＊ｐＢｕｃｋｅｔＬｉｓｔ）＝０
ａ）説明
このメソッドは、ポリシーマネージャディスパッチエンジン（ＣＰｏｌｉｃｙＭａｎａｇｅｒ）がシステム中のアクティビティについて優先度を計算するために呼び出す。バケットリストは、最初、このメソッドが第１のポリシーに対して呼び出されたときのシステム中のアクティビティすべてが入った単一のバケットを含む。次いでポリシーは、このバケットリストをコンパートメント化する。次いで、コンパートメント化されたバケットリストは、次の処理のためにポリシーマネージャディスパッチエンジンによって他のポリシーに渡される。
ｂ）パラメータ
ＰＢｕｃｋｅｔＬｉｓｔ：アクティビティオブジェクトを含むバケットのリストであるＣＢｕｃｋｅｔＬｉｓｔオブジェクトへのポインタ。
ｃ）戻り値
Ｓ＿ＯＫ　成功の場合。
Ｅ＿ＦＡＩＬ　そうでない場合。
【０１９２】
（ＣＣＯＮＦＬＩＣＴＰＯＬＩＣＹクラス中のメソッド（ＣＢａｓｅＰｏｌｉｃｙ中のメソッドに加えて））
１．ｖｉｒｔｕａｌ ＢＯＯＬ ＯｎＣｏｎｆｌｉｃｔ（ＣＡｃｔｉｖｉｔｙ＊ｐＣｏｎｆｌｉｃｔｉｎｇＡｃｔｉｖｉｔｙ，ＣＡｃｉｔｉｖｉｔｙＬｉｓｔ＊ｖｉｃｔｉｍＡｃｔｉｖｉｔｙＬｉｓｔ）＝０
ａ）説明
このメソッドは、リソースマネージャが現在のアクティビティ優先度で競合を解決することができないときに、ポリシーマネージャディスパッチエンジン（ＣＰｏｌｉｃｙＭａｎａｇｅｒ）によって呼び出される。このメソッドは、競合するアクティビティが犠牲者リスト中のすべてのアクティビティよりも重要か、またはそうでないかを決定することによって、競合を解決する。
ｂ）パラメータ
ＰＣｏｎｆｌｉｃｔｉｎｇＡｃｔｉｖｉｔｙ－予約中に競合を引き起こすアクティビティ。
ＶｉｃｔｉｍＡｃｔｉｖｉｔｙＬｉｓｔ－競合アクティビティによる予約を満たすためにリソースを放棄する必要のある犠牲者アクティビティのリスト。
ｃ）戻り値
ＴＲＵＥ　ＰＣｏｎｆｌｉｃｔｉｎｇＡｃｔｉｖｉｔｙが犠牲者リスト中のすべてのアクティビティよりも重要である場合。
ＦＡＬＳＥ　そうでない場合。
【０１９３】
（リソースマネージャＡＰＩの拡張）

```
1.HRESULT SetPolicyAttribute(IN HANDLE hActivity,
                             IN LPTSTR pAttrName,
                             IN DWORD dwType,
                             IN LPBYTE pData,
                             IN DWORD cbData)
```

ａ）説明
ポリシーデータベースにポリシー属性を追加するのに使用するメソッド。アプリケーションは、ＲＭＣｒｅａｔｅＡｃｔｉｖｉｔｙ（）から受け取ったアクティビティへのハンドルを使用して、ポリシー属性を指定することができる。
ｂ）パラメータ
ｈＡｃｔｉｖｉｔｙ－ＲＭＣｒｅａｔｅＡｃｔｉｖｉｔｙ（）によって返されるアクティビティへのハンドル。
ｐＡｔｔｒＮａｍｅ－終了ヌル文字を含めたポリシー属性の名前を含むバッファへのポインタ。
ｄｗＴｙｐｅ－指定の値に記憶されたデータタイプを示すＤＷＯＲＤコード。可能なタイプコードのリストについては、プラットフォームＳＤＫ文書中の「レジストリ値のタイプ」を参照されたい。
ｐＤａｔａ－指定の値についてのデータを含むバッファへのポインタ。
ｃｂＤａｔａ－ｐＤａｔａバッファのサイズ。

```
2.HRESULT GetPolicyAttribute(IN HANDLE hActivity,
                             IN LPTSTR pAttrName,
                             OUT LPDWORD lpType,
                             OUT LPBYTE pData,
                             OUT LPDWORD lpcbData)
```

ａ）説明
ポリシーデータベースに対してポリシー属性を取り出すのに使用するメソッド。
ｂ）パラメータ
ｈＡｃｔｉｖｉｔｙ－ＲＭＣｒｅａｔｅＡｃｔｉｖｉｔｙ（）によって返されるアクティビティへのハンドル。
ｐＡｔｔｒＮａｍｅ－終了ヌル文字を含めたポリシー属性の名前を含むバッファへのポインタ。
ｌｐＴｙｐｅ－指定の値に記憶されたデータタイプを示すコードを受け取る変数へのポインタ。
ｐＤａｔａ－指定の値についてのデータを受け取るバッファへのポインタ。
ｌｐｃｂＤａｔａ－返されるｐＤａｔａバッファのサイズ。
【０１９４】
（インテリジェントインタフェースコンポーネントを備えたアーキテクチャ）
図２に示すリソース管理アーキテクチャ１００では、アプリケーション３２が、タスクを完了するのに必要なリソースを要求できるほど十分にインテリジェントであると仮定している。しかし、アプリケーションが、それらに必要なリソースの完全なまたはいくらかの知識を有さないことが望ましい場合もある。さらに、リソース管理アーキテクチャ１００は、レガシーアプリケーションのある環境に実装される場合もある。このような状況では、レガシーアプリケーションが、リソース管理アーキテクチャの存在を知らなくてもこのようなアーキテクチャを利用できることが望ましい。
【０１９５】
図２０に、リソース管理アーキテクチャ２０００を示す。このアーキテクチャは図２と同様だが、アプリケーションとの間に入ってリソースを要求するインテリジェントインタフェースコンポーネント２００２を備える。インテリジェントインタフェースコンポーネント２００２は、カーネルレベルのコンポーネントでもよく（図示のように）、ユーザレベルのコンポーネントでもよい。
【０１９６】
インテリジェントインタフェースコンポーネント２００２があると、アプリケーション３２は、タスクを完了するのにどのリソースが必要かを知らなくてもよい。例えば、アプリケーション３２（１）がＴＶアプリケーションであるとする。ＴＶアプリケーションは、ストリーミングコンテンツをどのように表示装置上に表示するかについては関係があるが

、ストリーミングコンテンツがそもそもどのように受信されて表示されるかについてはほとんど関心がない場合がある。すなわち、ＴＶアプリケーションは、ＴＶ番組を受信して表示するために特定のチューナ、デコーダ、およびフィルタが必要であることを完全には意識していないことがある。

【０１９７】

このような場合、インタフェースコンポーネント２００２は、テレビジョンの表示など一般レベルのアクティビティにどのリソースが必要かを理解するように設計される。したがって、アプリケーション３２（１）が起動されたとき、アプリケーション３２（１）は、インタフェースコンポーネント２００２を呼び出してＴＶ番組の表示を要求する。インタフェースコンポーネント２００２は、リソースマネージャ１０２と対話して、アクティビティを生み出し、番組を表示するのに必要な同調リソース、デコードリソース、およびフィルタリングリソースの構成を構築する。この時点で、インタフェースコンポーネント２００２は本質的に、リソースの予約を要求するためにリソースマネージャに対処する際、図３のプロセスに関して先に述べたようにコンシューマとして振舞う。

【０１９８】

（スケジュールおよびステートレスプロバイダを備えたアーキテクチャ）

図２に示すリソース管理アーキテクチャ１００では、リソースプロバイダ１０４は、それがどのアクティビティに割り振られているか、割り振られたリソースの量、ならびに今割り振られているという概念について、何らかの認識を有することができる。しかし、必ずしもそうである必要はない。そうではなく、リソースプロバイダは、所有アプリケーションまたは割り振られたリソース量を知らないように構成することもできる。この情報は、リソースプロバイダ１０４のためにリソースマネージャ１０２によって維持される。この構成では、リソースプロバイダは「ステートレス」であるという。ステートレスなリソースプロバイダの使用が有益である理由の１つは、将来要求されるかもしれないあり得る構成をアーキテクチャが評価できるからである。

【０１９９】

図２１にリソース管理アーキテクチャ２１００を示すが、このアーキテクチャは、リソースプロバイダ２１０２（１）、２１０２（２）、．．．、２１０２（Ｐ）がステートレスである点で図２のアーキテクチャとは異なる。ステートレスなリソースプロバイダは、時間の概念なしで構成され、したがって、今要求されているのか将来要求されるのかについての認識を有さない。リソースプロバイダ２１０２は、所与の要求があった時にどのリソースがどれだけ使用されているかだけに関係し、この情報はリソースマネージャからリソースプロバイダに供給される。

【０２００】

アーキテクチャ２１００はまた、後の時点におけるリソースセットの割振りを予定するためのスケジューラ２１０４も備える。スケジューラ２１０４は、日時を追跡するためのカレンダーを備える。スケジューラ２１０４は、「もしも」シナリオを実行して、ステートレスリソースプロバイダ２１０２によって制御されるリソースが選択時に利用可能になるかどうかを決定するように構成される。例えば、スケジューラ２１０４が、午後８時などのプライムタイムのシステム使用を表す１つ以上のリソース構成のモックアップを作ると仮定する。スケジューラ２１０４は次いで、これらの構成にリソースを割り振ることができるかどうかリソースプロバイダ２１０２に尋ねる。プロバイダは時間の概念を有さず、その決定の根拠とすべきステートデータはリソースマネージャから渡されるので、プロバイダは単に、このような構成の集合を満たすことができるかどうかを示すだけである。

【０２０１】

（リソースマネージャＡＰＩ）

以下は、リソースマネージャについての例示的なＡＰＩである。以下に述べるＡＰＩ呼出しは、カーネルモードで利用可能である。リソースマネージャＡＰＩは、プロバイダインタフェース（セクションＡ）およびコンシューマインタフェース（セクションＢ）を含む。

【０２０２】

(A.プロバイダインタフェース)

プロバイダAPI呼出し

1.RmRegisterResource

a)プロトタイプ

```
NTSTATUS
    RmRegisterResource (
    IN LPCGUID ResourceType,
    IN PUNICODE_STRING ResourceName,
    IN PVOID ProviderContext,
    IN ULONG AccumulatorSize,
    IN RM_PROVIDER_CALLBACK CallbackTable,
    OUT PRM_HANDLE ResourceHandle
    );
```

b)説明

リソースプロバイダは、この関数を使用してリソースを登録する。各プロバイダは、サポートする各リソースタイプにつき1度、この関数を呼び出すべきである。

c)パラメータ

ResourceType:リソースのタイプを記述するリソースタイプGUIDへのポインタ。

ResourceName:リソースについてのユーザ可読の名前を指定するユニコードストリング。

ProviderContext:プロバイダコンテキストへのポインタ。このコンテキストポインタは、リソースマネージャからのすべてのコールバック中でプロバイダに返される。

AccumulatorSize:バイトで表したリソースアキュムレータバッファのサイズ。リソースマネージャがリソースアキュムレータを生み出す必要があるときは、リソースマネージャがリソースプロバイダのためにこのメモリ量を割り当てる。

CallbackTable:リソースマネージャがプロバイダにコールバックする際に使用

するコールバック関数へのポインタを含む構造体へのポインタ。この構造体は以下のように定義される。

```
typedef struct_RM_PROVIDER_CALLBACK {
        USHORT Version;
        USHORT Size;
        PRM_PROVIDER_ACCUMULATOR_ADD AccumulatorAdd;
        PRM_PROVIDER_ACCUMULATOR_INIT AccumulatorInit;
        PRM_PROVIDER_ACCUMULATOR_FREE AccumulatorFree;
        PRM_PROVIDER_ACCUMULATOR_COPY AccumulatorCopy;
        PRM_PROVIDER_RESERVE_NOTIFY NotifyOnReserve;
        PRM_PROVIDER_UNRESERVE_NOTIFY NotifyOnUnreserve;
        PRM_PROVIDER_REMOVE_NOTIFY NotifyOnRemove;
} RM_PROVIDER_CALLBACK, *PRM_PROVIDER_CALLBACK;
```

このコールバック構造体のフィールドは以下のとおりである。

Version:インタフェースのバージョン。プロバイダは、RmRegisterResourceを呼び出す前にこれをRM_VERSION_NUMに設定しなければならない。

Size:構造体のサイズ。プロバイダは、RmRegisterResourceを呼び出す前にこれを(RM_PROVIDER_CALLBACK)のサイズに設定しなければならない。

AccumulatorAdd:リソース属性をリソースアキュムレータに追加するために呼び出す、プロバイダへのコールバック。追加によってリソース割振り超過が生じない場合は、AccumulatorAddはアキュムレータを更新してSTATUS SUCCESSを返すべきである。追加によってリソース割振り超過が生じる場合は、プロバイダは、STATUS_RESOURCE_UNAVAILABLEを返し、バッファを未決定状態にしておくべきである。他のエラーがあれば、そのエラーは適切なエラーコードを返すべきである。

AccumulatorAdd関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    AccumulatorAdd (
    IN PVOID ProviderContext,
```

```
    IN PVOID AttributeBuffer,
    IN OUT PVOID Accumulator
    );
```

AccumulatorInit:リソースアキュムレータバッファを初期化するために呼び出す、プロバイダへのコールバック。NULLの場合は、リソースマネージャはバッファを0に初期化する。このコールバックは、リソースアキュムレータバッファが使用される前にリソースアキュムレータバッファに何らかの特別な初期化処理を施す機会をリソースプロバイダに与える。

AccumulatorInit関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    AccumulatorInit (
    IN PVOID ProviderContext,
    IN OUT PVOID Accumulator
    );
```

AccumulatorFree:リソースマネージャがアキュムレータバッファを開放する前に呼び出す、プロバイダへのコールバック。NULLの場合は、リソースマネージャはアキュムレータバッファを開放するときに何の特別な操作も行わない。このコールバックは、アキュムレータバッファ内のフィールドでポイントされるメモリを解放するかそうでない場合に割振り解除する機会を、リソースプロバイダに与える。リソースプロバイダは、リソースアキュムレータバッファを開放しようとすべきではない。

AccumulatorFree関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    AccumulatorFree (
    IN PVOID ProviderContext,
    IN OUT PVOID Accumulator
    );
```

AccumulatorCopy:あるリソースアキュムレータバッファを別のリソースアキュムレータバッファにコピーするために呼び出す、プロバイダへのコールバック。

NULLの場合は、リソースマネージャは単にバッファを直接コピーするだけである(例えばmemcpyを使用して)。このコールバックは、アキュムレータバッファ内のフィールドでポイントされるメモリをコピーする機会をリソースプロバイダに与える。

AccumulatorCopy関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    AccumulatorCopy (
    IN PVOID ProviderContext,
    IN PVOID SrcAccumulator,
    IN OUT PVOID DestAccumulator
    );
```

プロバイダは、DestAccumulatorを初期化されていないメモリとみなすべきである。具体的には、DestAccumulatorバッファは、AccumulatorCopy関数に渡される前にはどんなAccumulatorInit関数にも渡されない。

NotifyOnReserve:リソースが予約されたことをプロバイダに知らせるための、プロバイダへのコールバック。

NotifyOnReserve関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    NotifyOnReserve (
    IN PVOID ProviderContext,
    IN PVOID AttributeBuffer,
    );
```

プロバイダは、STATUS_RESOURCE_UNAVAILABLEを返すことにより、この呼出しに対して失敗することがある。例えば、デバイスが故障し、プロバイダがリソースを登録解除できる前にこの通知を受けた場合である。この場合、RMは予約をロールバックする。最も早い機会で、プロバイダはリソースカウントを更新する。

NotifyOnUnReserve:前に予約されたリソース量が未予約であることをプロバイダに知らせるための、プロバイダへのコールバック。

NotifyOnReserve関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    NotifyOnReserve (
    IN PVOID ProviderContext,
    IN PVOID AttributeBuffer,
    );
```

NotifyOnRemove:アクティビティ内のエラーのせいで、またはツリー中のより高レベルのリソースが除去されたせいで、またはプロバイダがRmRemoveResourceFromConfigurationを呼び出したせいで、前に追加されたリソースが除去されることをプロバイダに知らせるための、プロバイダへのコールバック。

NotifyOnRemove関数についてのプロトタイプは、以下のとおりである。

```
NTSTATUS
    NotifyOnRemove (
    IN PVOID ProviderContext,
    IN PVOID AttributeBuffer,
    );
```

プロバイダは、このコールバックを使用して、このリソースに関連するどんなデータ構造も解放すべきである。

注1:リソースマネージャは、NotifyOnRemove関数をRMRemoveResourceFromConfiguration内から呼び出すことになる。したがってプロバイダは、RMRemoveResourceFromConfigurationから戻った後は、この予約に関連するどんなデータ構造も解放しようとすべきではない。

注2:リソースマネージャは、NotifyOnRemove関数をRmAddResournceToConfiguration内の失敗の副作用として呼び出すことはない。したがって、何らかの理由でRmAddResourceToConfigurationが失敗した場合は、プロバイダは、このエラーの処理の一部としていかなるデータ構造のクリーンアップをも処理すべきである。

ResourceHandle:リソースマネージャへの他の呼出しで使用する、RmRegisterResourceから返されたハンドル。

d)戻り値

STATUS_SUCCESS:リソースの登録が成功した場合。
他の戻り値はどれもエラーを示す。
2.RmUnregisterResource
a)プロトタイプ

```
NTSTATUS
    RmUnregisterResource (
    IN RM_HANDLE ResourceHandle
    );
```

b)説明
リソースプロバイダは、そのリソースをリソースマネージャから登録解除する必要があるときに、この関数を呼び出すべきである。プロバイダは、この呼出しを行う前に、未処理のリソースがあればRmRemoveResourceFromConfigurationを使用してすべての構成から除去すべきである。プロバイダがこれを行わない場合、RMは、あたかもプロバイダがRmRemoveResourceFromConfigurationを呼び出したかのように自動的にすべてのリソースをプロバイダからパージする。
c)パラメータ
ResourceHandle:RmRegisterResourceから元々返されたハンドル。
d)戻り値
STATUS_SUCCESS:リソースの登録解除が成功した。
3.RmLockResource

```
NTSTATUS
    RmLockResource (
    IN RM_HANDLE ResourceHandle,
    );
```

a)説明
リソースプロバイダは、プロバイダが表すリソースの量が変化したときなどに、いずれかの内部状態を修正する前にこの関数を呼び出すべきである。これにより、リソースマネージャは、プロバイダのいかなる内部状態の変化とも同期することができる。

b)パラメータ
ResourceHandle:RmRegisterResourceから元々返されたハンドル。
c)戻り値
STATUS_SUCCESS:リソースのロックが成功した。
4.RmUnlockResource

```
NTSTATUS
    RmUnlockResource (
    IN RM_HANDLE ResourceHandle,
    IN BOOL Update
    );
```

a)説明
リソースプロバイダは、プロバイダが表すリソースの量が変化したときなどに、いずれかの内部状態の修正を完了する前にこの関数を呼び出すべきである。これにより、リソースマネージャは、リソース可用性の変化に応答してシステム中のアクティビティすべてのリソース使用を再計算することができる。
b)パラメータ
ResourceHandle:RmRegisterResourceから元々返されたハンドル。
Update:リソースの量が変化した場合、このパラメータをTRUEに設定すべきである。これは、システムのリソース使用を再計算するようにリソースマネージャに伝えることになる。
c)戻り値
STATUS_SUCCESS:リソースのロック解除が成功した。
5.RmAddResourceToConfiguration
a)プロトタイプ

```
NTSTATUS
    RmAddResourceToConfiguration (
    IN PRM_ACTIVITY Activity,
    IN PVOID Tag OPTIONAL,
    IN RM_HANDLE ParentId,
```

```
    IN RM_RESOURCE ResourceHandle,
    IN PVOID AttributeBuffer OPTIONAL,
    IN ULONG AttributeBufferSize,
    OUT PRM_HANDLE ResourceId
    );
```

b)説明

構成にリソースを追加する。

c)パラメータ

Activity:前に生み出されたアクティビティオブジェクト。

Tag:追加されつつある子を一意に識別/タグ付けするために親から供給されるポインタ。

ParentId:追加されつつあるリソースの親のリソースハンドル。

ResourceHandle:RmRegisterResourceから返されたハンドル。

AttributeBuffer:プロバイダだけに理解されるデータ(例えばリソース量)を含むバッファへのポインタ。AttribributeBufferSizeが0の場合、これはNULLになるはずである。

リソースマネージャは将来のコールバック関数すべてについてこのバッファの内部コピーを作成することに留意されたい。したがって、リソースプロバイダがこのバッファをそれ自体のスタック上に生み出すこと、および/またはRmAddResourceToConfigurationから戻った後でそれを廃棄することは許容される。しかしリソースマネージャは、この構造体のいずれかの内部フィールドにポイントされたデータをコピーすることはない。このようなデータはプロバイダによって維持されなければならず、いかなるプロセスコンテキスト内でもアクセス可能でなければならない。

AttributeBufferSize:バイトで表したAttributeBufferのサイズ。バッファにデータがない場合は0であるべきである。

ResourceId:このリソース割振りを表すためにこの呼出しによって返されるハンドル。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

6.RmRemoveResourceFromConfiguration

a)プロトタイプ

```
NTSTATUS
    RmRemoveResourceFromConfiguration (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceId
    );
```

b)説明

構成からリソース割振りを除去する。リソースがプロバイダに割り当てられている場合は、自動的に解放される。

c)パラメータ

Activity:前に生み出されたアクティビティオブジェクト。

ResourceId:RmAddResourceToConfigurationから返されたハンドル。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

7.RmSetResourceAttributes

a)プロトタイプ

```
NTSTATUS
    RmSetResourceAttributes (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceID,
    IN PVOID AttributeBuffer OPTIONAL,
    IN ULONG AttributeBufferSize
    );
```

b)説明

リソースに関する属性を変更する。

c)パラメータ

Activity:前に生み出されたアクティビティオブジェクト。

ResourceID:RmAddResourceToConfigurationから返されたハンドル。
AttributeBuffer:新しい属性を含むバッファへのポインタ。AttributeBufferSizeが0の場合は、NULLになるはずである。
AttributeBufferSize:バイトで表したバッファサイズ。
d)戻り値
STATUS_SUCCESSまたは適切なエラーコード。
8.RmgGetDefaultActivityAndConfiguration
a)プロトタイプ

```
NTSTATUS
    RmGetDefaultActivityAndConfiguration (
    OUT PRM_ACTIVITY *Activity,
    OUT PRM_HANDLE ConfigId
    );
```

b)説明
デフォルトのアクティビティおよび構成を返す。プロバイダは、レガシークライアントの場合など、リソースを追加すべきアクティビティおよび構成がわからない場合に、この呼出しを使用することができる。
c)パラメータ
Activity:デフォルトのアクティビティを返す。
ConfigId:デフォルトの構成を返す。
d)戻り値
STATUS_SUCCESSまたは適切なエラーコード。
9.RmGetAttributeBuffer
a)プロトタイプ

```
NTSTATUS
    RmGetAttributeBuffer (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceId,
    IN PVOID *Buffer


    );
```

b)説明
リソース記述子に関連する属性バッファを得る。
c)パラメータ
Activity:前に生み出されたアクティビティへのポインタ。
ResourceID:上の引数によるアクティビティ内のリソース記述子へのハンドル
。
Buffer:属性バッファへのポインタを受け取る変数。
d)戻り値
STATUS_SUCCESSまたは適切なエラーコード。

【０２０３】

(B.コンシューマインタフェース)

コンシューマAPI呼出し

1.RmCreateActivity

a)プロトタイプ

```
NTSTATUS
    RmCreateActivity (
    OUT PRM_ACTIVITY ActivityObject,
    IN ACCESS_MASK DesiredAccess,
    IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN REFGUID TypeGuid
    );
```

b)説明

この関数は新しいアクティビティオブジェクトを生み出す。

c)パラメータ

ActivityObject:生み出されるアクティビティオブジェクトへのポインタがここに返される。

DesiredAccess:アクセス権。例えばACTIVITY_QUERY_STATEやACTIVITY_MODIFY_STATE。

ObjectAttributes:名前やセキュリティなどの標準的なオブジェクト属性を供給する。

TypeGuid:アクティビティのタイプ。事前定義済みタイプのセットのうちの1つ。アクティビティタイプを使用して、他のアクティビティに対する相対的なアクティビティの優先度を決定するのを補助することができる。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

e)注

単一のリソースコンシューマが複数のアクティビティを生み出すことができる。

2.RmReserveResources

a)プロトタイプ

```
NTSTATUS
    RmReserveResources (
    IN PRM_ACTIVITY Activity,
    IN OUT PRM_ACTIVITY_STATUS ActivityStatus,
    IN PKEVENT Event,
    IN RM_HANDLE DesiredConfigId,
    );
```

b)説明

RMは、アクティビティ内の構成のためのリソースを予約する。選択される構成は、リソースの可用性およびアクティビティの相対的な優先度に基づく。呼出し元は、任意選択で有効なConfigIdを供給することができ、その場合、RMはその構成を予約することを試みるだけであり、そうでない場合は失敗する。入力時にアクティビティがすでに十分な構成を有する場合(すなわちリソースが構成に割り当てられている場合)、RMはこれを、より望ましい構成が利用可能かどうか調べる要求であると解釈する。RMは、少なくとも既存の構成がリターン時にまだ利用可能であることを保証する(優先度のより高いアクティビティがリソースを必要としているなど、他の理由で既存の構成が立ち退かされていないと仮定して)。

この呼出しは非同期である。ActivityStatusおよびEventのパラメータを使用することにより、この呼出し元は、要求を満たすことができるかどうかをRMが決定するまで待機することができる。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ActivityStatus:この構造体は、要求が完了した後でRMが要求の状態を返すのに使用する。この構造体は以下のように定義される。

```
typedef struct_RM_ACTIVITY_STATUS
{
        NTSTATUS Status;
        ULONG Information;
```

} RM_ACTIVITY_STATUS, *PRM_ACTIVITY_STATUS;

RmReserveResourcesと共に使用されるとき、Statusフィールドは、操作結果を保持するのに使用され、Informationフィールドは、RMが満たした構成IDを受け取る。Informationフィールドは、Statusの値がSTATUS_SUCCESSである場合のみ有効である。

Event:RMがすぐに予約を完了することができない場合、呼出し元はこのイベントを待機して、操作が完了したときを決定すべきである。

DesiredConfigId:呼出し元は、このパラメータを具体的な構成IDで初期化することにより、そのIDの予約を選択することができる。そうでない場合は0で初期化すべきである。

d)戻り値

STATUS_SUCCESS:RMがアクティビティ内のいずれかの構成を満たすことができる場合。

STATUS_RESOURCE_UNAVAILABLE:優先度のより高い他のアクティビティにリソースが割り当てられているので、現在どの構成も満たすことができない。

STATUS_PENDING:RMがまだ操作を完了することができない場合。この場合、呼出し元は、操作が完了するまでEventハンドルを待機すべきである。

3.RmUnreserveResources

a)プロトタイプ

```
NTSTATUS
    RmUnreserveResources (
    IN PRM_ACTIVITY Activity
    );
```

b)説明

アクティビティ内の構成に関連するリソースを予約解除する。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

d)戻り値

STATUS_SUCCESSまたはエラーコード。

4.RmGetActivityStatus

a)プロトタイプ

```
NTSTATUS
    RmGetActivityStatus (
    IN PRM_ACTIVITY Activity,
    IN OUT PRM_ACTIVITY_STATUS ActivityStatus,
    IN PKEVENT Event
    );
```

b)説明

アクティビティの状態の変化に関する情報を返す。この呼出しは非同期である。ActivityStatusおよびEventのパラメータを使用することにより、この呼出し元は、アクティビティの状態が変化したとRMが決定するまで待機することができる。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ActivityStatus:この構造体は、要求が完了した後でRMが要求の状態を返すのに使用する。この構造体は以下のように定義される。

```
typedef struct_RM_ACTIVITY_STATUS
{
      NTSTATUS Status;
      ULONG Information;
} RM_ACTIVITY_STATUS, *PRM_ACTIVITY_STATUS;
```

RmGetActivityStatusと共に使用されるとき、Statusフィールドは、操作結果を保持するのに使用され、Informationフィールドは理由を受け取る。例として、RM_RELEASE_CONFIGURATIONやRM_BETTER_CONFIGURATIONなどが挙げられる。Informationフィールドは、Statusの値がSTATUS_SUCCESSである場合のみ有効である。

Event:保留中の状態が現在ない場合、呼出し元はこのイベントを待機して、新しい状態情報が入手可能になったときを決定すべきである。

d)戻り値
STATUS_SUCCESSまたは適切なエラーコード。
5.RmCancelRequest
a)プロトタイプ

```
NTSTATUS
    RmCancelRequest(
    IN PRM_ACTIVITY Activity,
    );
```

b)説明
この呼出しは、任意の保留中の非同期呼出しをキャンセルするのに使用することができる。現在、RmReserveResourcesおよびRmGetResourceStatusだけが非同期として定義されている。
この呼出しは、RmCancelRequestを呼び出す同じスレッドから開始された任意の保留中RM要求をキャンセルする。RMがトランザクションをキャンセルするときは、RM_ACTIVITY_STATUS構造体のStatusフィールドをSTATUS_CANCELLEDに設定し、対応するイベントを信号で知らせる。
トランザクションをキャンセルした後、呼出し元は、RM_ACTIVITY_STATUS構造体のStatusフィールドを常にチェックして、トランザクションが実際にキャンセルされたかどうか、またはトランザクションがキャンセルされる前にうまく完了したかどうかを検証すべきである。
c)パラメータ
Activity:前に生み出されたアクティビティのハンドル。
d)戻り値
STATUS_SUCCESS:RMが指定のタイプのトランザクションをすべてキャンセルした。
6.RmCreateConfiguration
a)プロトタイプ

```
NTSTATUS
    RmCreateConfiguration (
```

```
    IN PRM_ACTIVITY Activity,
    IN ULONG Merit,
    OUT PRM_HANDLE ConfigId
    );
```

b)説明

指定のアクティビティ内に構成を生み出す。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

Merit:このアクティビティ内の他の構成に対して相対的なこの構成の重要性。

ConfigId:生み出される構成へのハンドル。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

7.RmRemoveConfiguration

a)プロトタイプ

```
NTSTATUS
    RmRemoveConfiguration (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ConfigId
    );
```

b)説明

指定のアクティビティから構成を除去する。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ConfigId:除去される構成へのハンドル。予約されているリソースがあれば、RMは自動的にクリーンアップする。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

8.RmGetResourceParent

a)プロトタイプ

```
NTSTATUS
    RmGetResourceParent (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceIdIn,
    OUT PRM_HANDLE ResourceIdReturned
    );
```

b)説明

ResourceIdInの親ResourceIdを返す。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ResourceIdIn:ResourceIdへのハンドル。

ResourceIdReturned:ResourceIdへのポインタ。得られるResourceIdがこの位置に返される。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

9.RmGetResourceChild

a)プロトタイプ

```
NTSTATUS
    RmGetResourceChild (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceIdParent,
    IN RM_HANDLE ResourceIdIn,
    OUT PRM_HANDLE ResourceIdReturned
    );
```

b)説明

ResourceIdInの第1の子ResourceIdを返す。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ResourceIdParent:ResourceIdまたはConfigurationIdへのハンドル。

ResourceIdIn:ResourceIdへのハンドル。1に設定された場合、関数は第1の子リソースを返す。
ResourceIdReturned:ResourceIdへのポインタ。得られるResourceIdがこの位置に返される。
d)戻り値
STATUS_SUCCESSまたは適切なエラーコード。
10. RmGetResourceInformation
a)プロトタイプ

```
NTSTATUS
    RmGetResourceInformation (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceId,
    IN RM_RESOURCE_INFO_TYPE ResourceInfoType,
    IN ULONG AvailableBufferSize,
    OUT ULONG *RequiredBufferSize,
    OUT PVOID *ResourceInfo
    );
```

b)説明
指定のResourceIdに関する情報を返す。
c)パラメータ
Activity:前に生み出されたアクティビティのハンドル。
ResourceId:ResourceIdまたはConfigurationIdへのハンドル。
ResourceInfoType:どのタイプの情報を返すかを指定するenum値。現在定義されている値およびそれらが返す対応バッファは、以下のとおりである。
ResourceInfo_Default
バッファ中に以下の構造体を返す。

```
typedef struct_RM_RESOURCE_INFO_DEFAULT
{
        BOOL FailedReservation;          //このリソースが、その最後の予
```

```
約を失敗した (This resource failed its last reservation)
        BOOL DescendantFailedReservation; //子孫が、その最後の予約を失敗
した (A descendant failed its last reservation)
        BOOL PeerFailedReservation;      //同じプロバイダの仲間が、最後
の予約を失敗した (Peer of same provider failed last reservation)
        GUID ResourcePoolGUID;           //リソースGUID (Resource GUID
)
        RM_HANDLE ResourcePoolHandle;    //プロバイダのハンドル (Handle
 of provider)
        ULONG TagLength;                 //タグバッファのサイズ (Size of
 tag buffer)
} RM_RESOURCE_INFO_DEFAULT;
  ResourceInfo_Tag
  リソースが追加されたときに元々指定されたタグを返す。
  ResourceInfo_Amount
  バッファ中に以下の構造体を返す。
typedef struct_RM_RESOURCE_INFO_AMOUNT
{
        ULONG AmountReserved;
        ULONG AmountInSystem;
        ULONG AmountAvailable;
        ULONG AmountUsed;
        ULONG AmountUsedPeak;
        WCHAR AmountLabel?32?;
        WCHAR ProviderDescription?128?;
} RM_RESOURCE_INFO_AMOUNT;
  AvailableBufferSize:バイトで表したバッファサイズ。
  RequiredBufferSize:バイトで表した、バッファ中に返されるデータのサイズ
```

ResourceInfo:RMが要求のデータを記憶することになるバッファへのポインタ

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

11.RmGetLastAccumulator

a)プロトタイプ

```
NTSTATUS
    RmGetLastAccumulator(
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ProviderHandle,
    OUT PVOID *AccumulatorBuffer
    );
```

b)説明

指定されたアクティビティのリソースのいずれかを追加する前の、最後のアキュムレータバッファを返す。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ProviderHandle:RmRegisterResourceから返されたハンドル。

AccumulatorBuffer:リターンが成功したときにアキュムレータバッファへのポインタがここに記憶される。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

e)コメント

この呼出しは、リソース不足のせいでアクティビティが通常なら立ち退かされることになる場合に、リソースプロバイダが再度アクティビティ内のリソースのバランスをとることができるようにするために、RmGetResourcePeerと共に使用することができる。

12.RmGetResourcePeer

a)プロトタイプ

```
NTSTATUS
    RmGetResourcePeer (
    IN PRM_ACTIVITY Activity,
    IN RM_HANDLE ResourceProvider,
    IN RM_HANDLE ResourceIdIn,
    OUT PRM_HANDLE ResourceIdReturned
    );
```

b)説明

指定されたアクティビティ内の、指定されたリソースプロバイダを有する次のリソースのResourceIdを返す。

c)パラメータ

Activity:前に生み出されたアクティビティのハンドル。

ResourceProvider:リソースプロバイダへのハンドル。

ResouceIdIn:ResouceIdへのハンドル。このパラメータが1に設定された場合、関数は指定のプロバイダに関連する第1のリソースを返す。

ResourceIdReturned:ResouceIdへのポインタ。得られるResourceIdがこの位置に返される。

d)戻り値

STATUS_SUCCESSまたは適切なエラーコード。

e)コメント

この呼出しを実装すると、以下のようなRmGetResourceInformationの変更が引き起こされることがある。

1.リソースに関連する属性バッファを返すためにResourceInfo_AttributesのResourceInfoTypeを追加する。

2.RM_RESOURCE_INFO_DEFAULT構造体にULONG AttributeBufferLengthフィールドを追加する。

【０２０４】

(結び)

以上の説明では、構造上の特徴、および／または方法の動作に特有の言葉を使用したが、頭記の特許請求の範囲に定義する本発明は、述べた具体的な特徴または動作に限定されるものではないことを理解されたい。そうではなく、これらの具体的な特徴および動作は、本発明を実施する例示的な形として開示する。

【図面の簡単な説明】

【図１】

一実施形態による、娯楽コンピューティングユニットの形をとる例示的なコンピューティングユニットのブロック図である。

【図２】

図１のコンピューティングユニットによって実現される例示的なリソース管理アーキテクチャのブロック図である。

【図３】

図２のリソース管理アーキテクチャによって実現される例示的なリソース管理方法のステップを記述する流れ図である。

【図４】

述べた実施形態による例示的なリソース予約プロセスを示すブロック図である。

【図５】

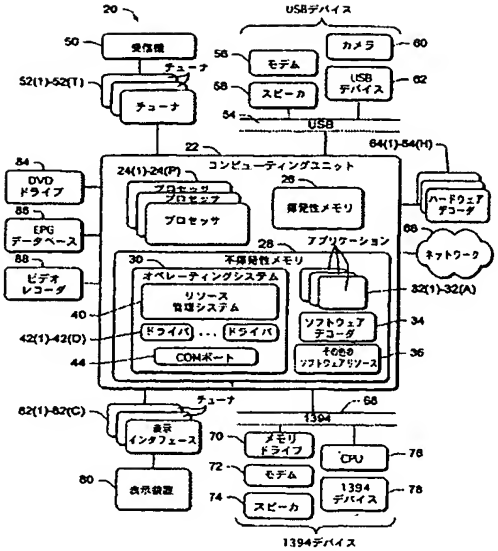
述べた実施形態による、リソース予約プロセスで利用される例示的な種々の構造を示すブロック図である。

【図６】

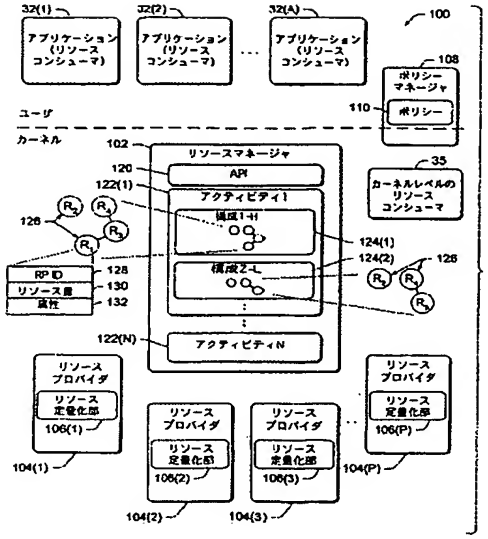
述べた実施形態による、優先度ベースのプリエンプトを用いたリソース割振り方法におけるステップを記述する流れ図である。

【図７】
述べた実施形態による、優先度ベースのプリエンプトを用いたリソース割振りについてのシナリオを示すブロック図である。
【図８】
述べた実施形態による、優先度ベースのプリエンプトを用いたリソース割振りについてのシナリオを示すブロック図である。
【図９】
述べた実施形態による、優先度ベースのプリエンプトを用いたリソース割振りについてのシナリオを示すブロック図である。
【図１０】
述べた実施形態による、優先度ベースのプリエンプトを用いたリソース割振りについてのシナリオを示すブロック図である。
【図１１】
述べた実施形態による、より好ましい構成からより好ましくない構成に動的にダウングレードする方法におけるステップを記述する流れ図である。
【図１２】
述べた実施形態による、より好ましくない構成にダウングレードする方法におけるシナリオの１つを示すブロック図である。
【図１３】
述べた実施形態による、より好ましくない構成からより好ましい構成に動的にアップグレードする方法におけるステップを記述する流れ図である。
【図１４】
述べた実施形態による、より好ましい構成にダウングレードする方法におけるシナリオの１つを示すブロック図である。
【図１５】
構成を構築する方法におけるシナリオの１つを示すブロック図である。
【図１６】
述べた実施形態による、構成を構築する方法におけるステップを記述する流れ図である。
【図１７】
述べた実施形態による、エラー報告方法におけるステップを記述する流れ図である。
【図１８】
例示的なポリシー管理アーキテクチャのブロック図であって、図２のリソース管理アーキテクチャを用いて実現されるポリシーマネージャをさらに示す図である。
【図１９】
図１８のポリシー管理アーキテクチャによって実現される例示的なポリシー管理方法におけるステップを記述する流れ図である。
【図２０】
図２と同様だがインテリジェントインタフェースコンポーネントをさらに備える、別の例示的なリソース管理アーキテクチャのブロック図である。
【図２１】
図２と同様だがスケジューリングコンポーネントをさらに備える、別の例示的なリソース管理アーキテクチャのブロック図である。

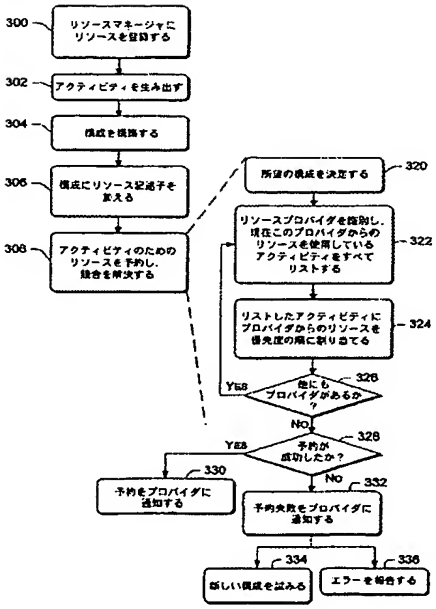
【図１】


20
USBデバイス
50
受信機
56
モデム
カメラ
60
チューナ
52(1)-52(T)
チューナ
58
スピーカ
USB
デバイス
62
54
USB
22
64(1)-64(H)
コンピューティングユニット
24(1)-24(P)
84
DVD
ドライブ
プロセッサ
プロセッサ
プロセッサ
揮発性メモリ
26
ハードウェア
デコーダ
85
EPG
データベース
66
ネットワーク
28
アプリケーション
88
ビデオ
レコーダ
30
不揮発性メモリ
オペレーティングシステム
32(1)-32(A)
リソース
管理システム
40
ドライバ
ドライバ
ソフトウェア
デコーダ
34
42(1)-42(D)
COMポート
その他の
ソフトウェアリソース
36
44
チューナ
62(1)-62(C)
68
1394
表示
インタフェース
70
メモリ
ドライブ
CPU
78
72
モデム
80
表示装置
1394
デバイス
78
74
スピーカ
1394デバイス

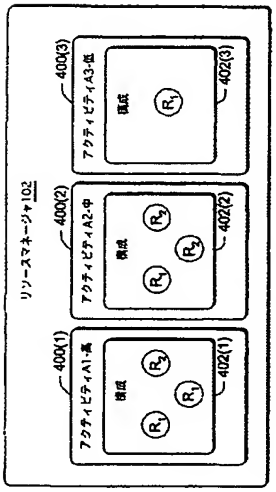
【図２】


32(1)
32(2)
32(A)
100
アプリケーション
(リソース
コンシューマ)
アプリケーション
(リソース
コンシューマ)
アプリケーション
(リソース
コンシューマ)
108
ポリシー
マネージャ
110
ポリシー
ユーザ
カーネル
102
リソースマネージャ
120
API
35
カーネルレベルの
リソース
コンシューマ
122(1)
アクティビティ1
構成1-H
126
124(1)
124(2)
126
構成2-L
RFID
128
リソース量
130
属性
132
122(N)
アクティビティN
リソース
プロバイダ
リソース
定量化部
106(1)
104(1)
リソース
プロバイダ
リソース
定量化部
106(2)
104(2)
リソース
プロバイダ
リソース
定量化部
106(3)
104(3)
リソース
プロバイダ
リソース
定量化部
106(P)
104(P)

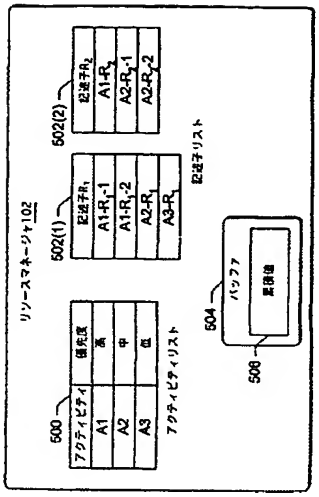
【図３】


300
リソースマネージャに
リソースを登録する
302
アクティビティを生み出す
304
構成を模索する
306
構成にリソース記述子を
加える
308
アクティビティのための
リソースを予約し、
競合を解決する
320
所望の構成を決定する
322
リソースプロバイダを指定し、
現在このプロバイダからの
リソースを使用している
アクティビティをすべて
リストする
324
リストしたアクティビティに
プロバイダからのリソースを
優先度の順に割り当てる
326
Yes
他にも
プロバイダがあるか
？
NO
328
Yes
予約が
成功したか？
No
330
予約をプロバイダに
通知する
332
予約失敗をプロバイダに
通知する
334
新しい構成を試みる
336
エラーを報告する

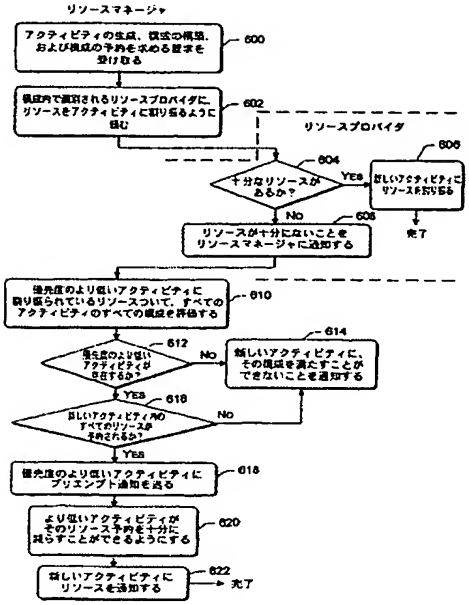
【図４】


リソースマネージャ102
400(3)
アクティビティA3-低
構成
402(3)
400(2)
アクティビティA2-中
構成
402(2)
400(1)
アクティビティA1-高
構成
402(1)

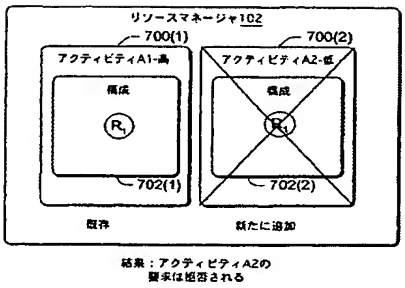
【図５】



【図６】

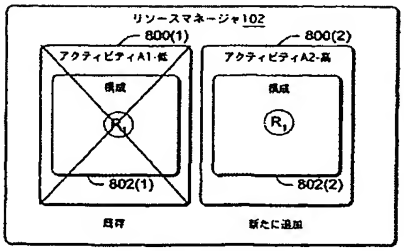


【図７】



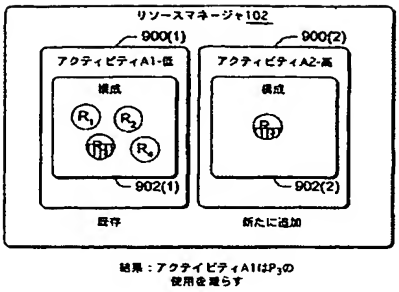
結果：アクティビティA2の要求は拒否される

【図８】

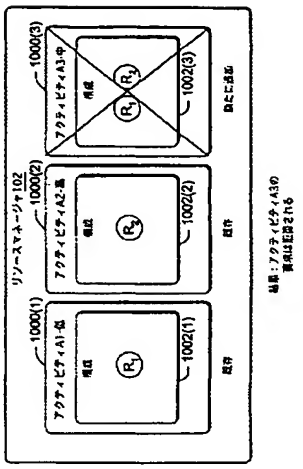


結果：アクティビティA1はプリエンプトされる

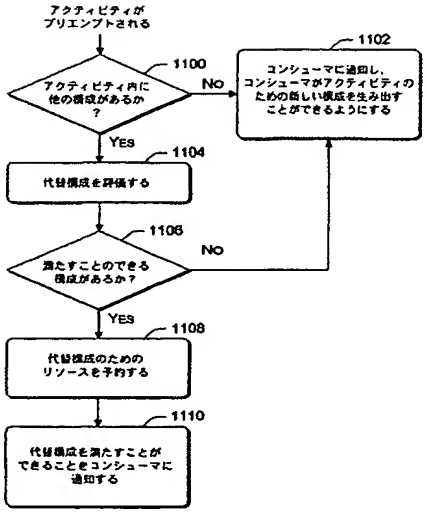
【図９】



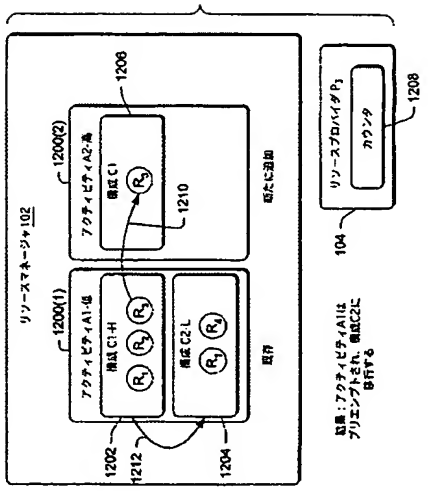
【図１０】



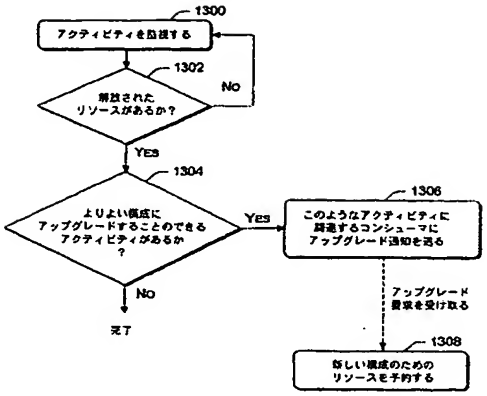
【図１１】



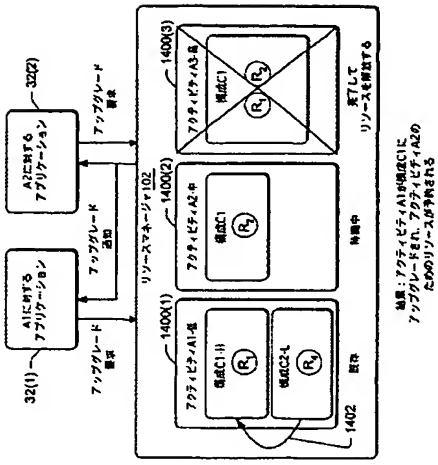
【図１２】



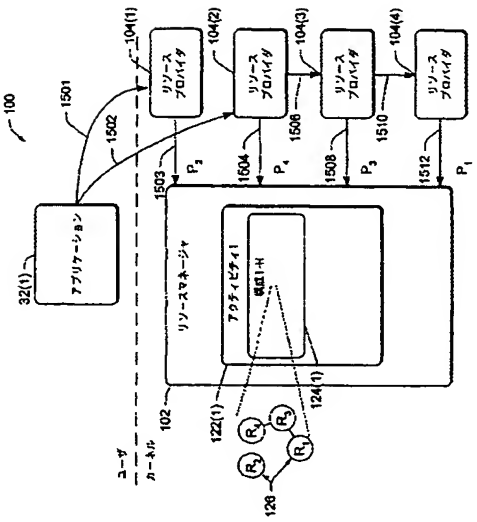
【図１３】



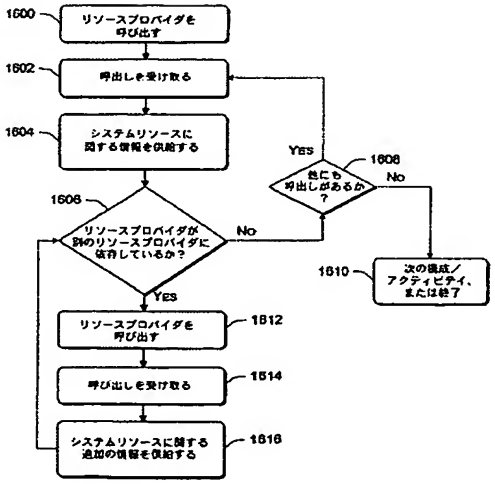
【図１４】



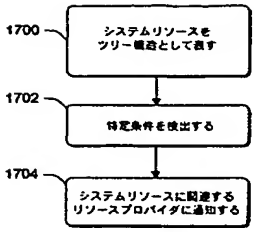
【図１５】



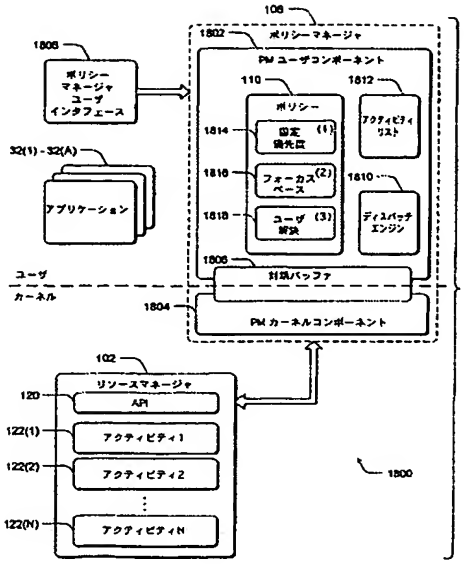
【図１６】



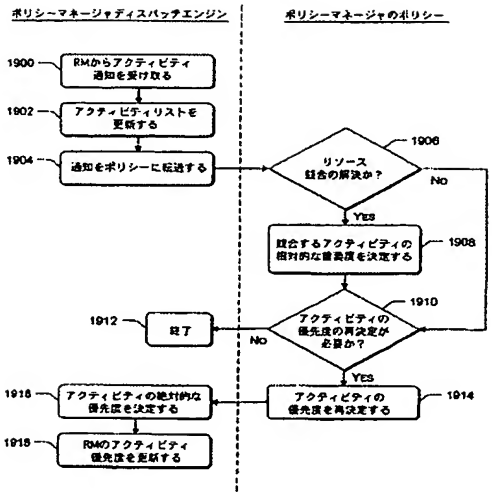
【図１７】



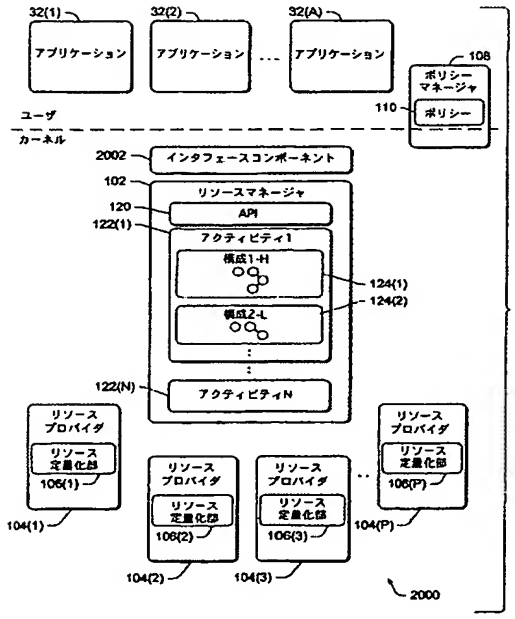
【図１８】



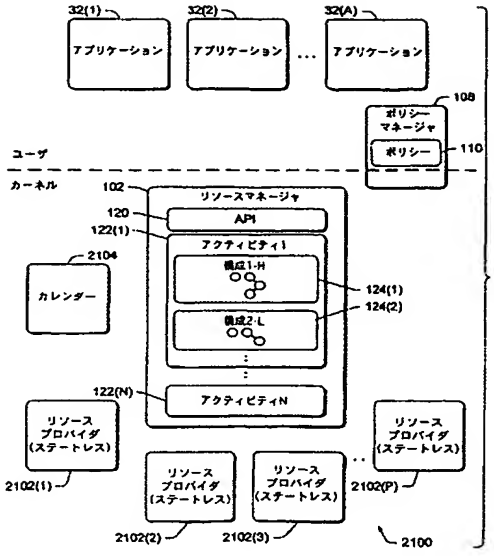
【図１９】



【図２０】



【図２１】


32(1)
32(2)
32(A)
アプリケーション
アプリケーション
アプリケーション
108
ポリシー
マネージャ
ポリシー
110
ユーザ
カーネル
102
リソースマネージャ
120
API
122(1)
アクティビティ1
2104
カレンダー
構成1-H
構成2-L
124(1)
124(2)
122(N)
アクティビティN
リソース
プロバイダ
(ステートレス)
2102(1)
リソース
プロバイダ
(ステートレス)
2102(2)
リソース
プロバイダ
(ステートレス)
2102(3)
リソース
プロバイダ
(ステートレス)
2102(P)
2100